VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Síťování ve virtualizovaném prostředí Linuxu

Linux Virtualized Container Environment in the Networking

2020 Bc. Antonín Salzmann

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Antonín Salzmann**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Síťování ve virtualizovaném kontejnerovém prostředí linuxu
Linux Virtualized Container Environment in the Networking

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem diplomové práce je prostudovat, navrhnout a ověřit jednotlivé možnosti síťování ve virtualizovaném protředí kontejnerů.

1. Popis a vlastnosti kontejnerové virtualizace.
2. Síťování pomocí kontejnérů v linuxu.
3. Návrh zapojení a ověření funkčnosti.
4. Testování, měření a vyhodnocení výkonnosti.

Seznam doporučené odborné literatury:

[1] IVANOV, Konstantin *Containerization with LXC* Packt Publishing 2017, ISBN-13: 978-1785888946
[2] SHASHANK, Mohan, Jain *Linux Containers and Virtualization - A kernel perspective* Independently published 2019, ISBN-13: 978-1080299423

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Nevlud**

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 15. května 2020

Salzmann .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 15. května 2020

Sekmann .....

Rád bych poděkoval za cenné rady, odborný přístup, vstřícnost a trpělivost vedoucímu práce, panu Ing. Pavlu Nevludovi. Velice si vážím času, který mi věnoval při realizace této práce.

A dále bych rád poděkoval své rodině za podporu, nejen při psaní této práce, ale po celou dobu mého studia.

Abstrakt

Tato diplomová práce se zabývá popisem druhů virtualizaci, především pak virtualizací kontejnerovou a jejím síťování. Pro kontejnerovou virtualizaci používám nástroj LXD, což je nástavba nad nástrojem LXC. Využívám protokolu IPv6, jelikož se na něj dříve nebo později bude muset z protokolu IPv4 přejít, nejen kvůli nedostatku adres, ale i dalším nedostatkům protokolu.

V první části nejprve teoreticky popisuji jednotlivé techniky virtualizace, poté se zaměřuji podrobněji na kontejnerovou virtualizaci a její komponenty - cgroup a namespaces.

V druhé části práce ukazuji možnosti a ukázky virtuálního síťování v prostředi Linuxu.

V praktické části pak ukazuji možnosti síťování a ukázky konfigurace v prostředí LXD.

Klíčová slova: Linux; virtualizace; LXC; LXD; IPv4; IPv6; kontejner; síťování; cgroups; namespaces

Abstract

This thesis deals with the description of virtualization types, especially with container virtualization and its networking features. I used LXD tool for container virtualization, which is extension of tool LXC. I am also using protocol IPv6, because sooner or later we will have to switch to it from protocol IPv4, not only due to the lack of IPv4 addresses, but also to other issues related to to IPv4 protocol.

In the first part, I describe various types of virtualization techniques, then I focus on explaining container virtualization in depth and its componenets - cgroups and namespaces.

In the second part of the thesis I show the possibilities and examples of virtual networking in the Linux environment itself.

And in the last and practical part I show the possibilities of networking and various configuration examples in the LXD environment.

Keywords: Linux; virtualization; LXC; LXD; IPv4; IPv6; container; networking; cgroups; namespaces

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CPU	–	Central Processing Unit
DHCP	–	Dynamic Host Configuration Protocol
DNS	–	Domain Name Server
EUI-64	–	Extended Unique Identifier
GID	–	Group Identifier
IPC	–	Interprocess Communication
IPV4	–	Internet Protocol version 4
IPV6	–	Internet Protocol version 6
LTS	–	Long Term Support
LXC	–	Linux Containers
mnt	–	mount
NAT	–	Network Address Translation
net	–	network
NIC	–	Network Interface Card
ns	–	namespace
PID	–	Process Identifier
RAM	–	Random Access Memory
SLAAC	–	Stateless Address Autoconfiguration
SSD	–	Solid State Drive
UID	–	User Identifier
UTS	–	UNIX Time Sharing
VEPA	–	Virtual Ethernet Port Aggregator
VM	–	Virtual Machine
VMM	–	Virtual Machine Monitor

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Zpět v 60. letech minulého století byly počítače vzácným a drahým zbožím. I jen jejich samotný pronájem stál měsíčně velké peníze, což je stavělo mimo dosah pro mnoho společností. A jelikož se říká, že nutnost je matkou vynálezu, tak ani historie počítačů nebyla výjimkou.

Prvotní počítače byly typicky určeny pro specifické úlohy, které mohly trvat dny či dokonce i týdny, což byl důvod, jež vedl během 60. a 70. let k vývoji virtualizace. Podnět pro tento vývoj byla potřeba sdílet prostředky počítače mezi více uživateli zároveň ve stejný čas.

S nástupem střediskových (sálových) počítačů jsme mohli zahlédnout první náznaky toho, co dnes nazýváme virtualizací. Během oněch 60. let byly počítačové terminály připojeny k jednomu sálovému počítači (mainframe), což umožnilo provádět výpočetní práci a řídit veškeré zpracování z jednoho místa, takže pokud by například jeden terminál postihla porucha, uživatel by stále mohl jít k terminálu jinému, přihlásit se zde k počítači a stále mít přístup k všem svým souborům.

Nicméně to mělo své nevýhody. Například, pokud by selhal samotný počítač, systém by byl nedostupný pro všechny bez rozdílu. Problémy, jako je tento jasně ukázaly, že počítače musí být odděleny nejen pro jednotlivé uživatele, ale také i pro samotné systémové procesy.

V roce 1979 se postoupilo zase o krok kupředu k vytvoření sdílených, ale přesto izolovaných prostředí s vytvořením nástroje ***chroot***. Tento nástroj dokázal změnit zjevný kořenový adresář pro daný běžící proces a všechny jeho potomky. To umožnilo izolovat systémové procesy do jejich vlastních segregovaných souborových systémů, aby se mohlo provádět testování, bez dopadu na globální systémové prostředí.

Další pokrok přinesl americký vědecký pracovník William Cheswick, když v roce 1991 vytvořil honeypot pro monitorování crackerů. Jeho řešení bylo použití upraveného chroot prostředí. Výsledkem jeho práce je, co dnes nazýváme ***jail***. V roce 2000 byl příkaz jail implementován do operačního systému ***FreeBSD***, jež byl podobný příkazu chroot, ale umožnil dodatečné sandboxové funkce pro izolaci souborových systémů, uživatelů, síťových nastavení, atd. FreeBSD jail umožnil přiřadit IP adresu, konfigurovat software a dělat úpravy každému takovému prostředí. Mělo to však své problémy, jelikož aplikace uvnitř byly limitovýny ve své funkčnosti.

V roce 2004 vypustil Solaris své kontejnery, které vytvořily plná aplikační prostředí díky použití ***Solaris Zones***. Tyto zóny poskytují aplikacím prostor pro uživatele, procesy a souborové systémy spolu s přístupem k hardwaru. Nicméně aplikace mohly vidět jen to, co bylo v jejich vlastní zóně.

Dále posunuli vývoj vědci z Googlu, když roku 2006 oznámili spuštění jejich vlastních kontejnerů navržených pro izolování procesů a limitování systémových zdrojů pro ně. O rok později tyto kontejnery přejmenovali na cgroups (control groups), aby se nezaměňovaly se slovem kontejner (container). Cgroups byly poté implementovány do Linuxového jádra (od verze 2.6.24), což vedlo ke vzniku projektu LXC. LXC poskytuje virtualizaci na úrovni operačního systému umožněním běhu mnoha izolovaných Linuxových prostředí na sdíleném Linuxovém jádře. Každý z těchto kontejnerů má vlastní procesový a síťový prostor. [**5**]

A právě na projektu LXC je postaven projekt LXD, kterému se budu v této diplomové práci věnovat. V teoretické části práce nejprve popíši jednotlivé typy virtualizací, především však virtualizace kontejnerovou, dále možnosti konfigurace virtuálních síťových rozhraní v Linuxu a na závěr v praktické části ukáži možnosti síťování v kontejnerovém prostředí LXD.

2 Virtualizace

Virtualizace je proces běhu virtuální instance počítačového systému ve vrstvě oddělené od reálného fyzického hardwaru. Nejčastěji se jedna o současný běh více operačních systémů v rámci jednoho počítače. Aplikacím, běžícím ve VM, se zdá, že běží ve svém vlastním vyhrazeném počítači, kde operační systém, knihovny a další programy jsou jedinečné pro daný hostovaný systém a nejsou připojeny k hostitelskému systému, který je pod ním.

Důvodů, proč se virtualizace v oblasti výpočetní techniky hojně využívá, je celá řada. Některé jsem již zmínil v úvodu práce. Pro běžné uživatele stolních počítačů je nejčastějším využitím běh aplikací různých operačních systémů bez nutnosti mezi danými operačními systémy přepínat, či je mít vůbec na fyzickém hardwaru nainstalované. Pro administrátory serverů nabízí virtualizace také možnost provozovat různé operační systémy, ale spíše je pro ně důležitější možnost rozdělení systému do menších částí, což umožňuje, aby byl server využíván efektivně různými uživateli s různými potřebami. Dále také poskytuje izolaci, držením aplikací bezpečně uvnitř VM před procesy probíhajícími na jiném VM, která se nachází na stejném hostiteli.

2.1 Typy virtualizace

Nejčastějšími způsoby, jak vytvořit virtuální stroje jsou: plná virtualizace, para-virtualizace a virtualizace na úrovni operačního systému. Všechny typy sdílejí několik společných rysů. Fyzický stroj se nazývá hostující systém - hostitel (angl. host) a virtuální stroj se nazývá hostovaný systém (angl. guest). VM se chovají jako ty fyzické. Každý typ však používá jiný přístup k alokování prostředků fyzického stroje k potřebám těch virtuálních.

2.1.1 Plná virtualizace

Plná virtualizace používá speciální typ softwaru zvaný hypervizor. Hypervizor pracuje přímo s procesorem a diskovým prostorem fyzického stroje. Slouží jako platforma pro operační systémy virtuálních strojů. Hypervizor udržuje každý virtuální stroj jako zcela nezávislý, jež neví o ostatních virtuálních strojích běžících na stejném fyzickém počítači. Každý tento virtuální stroj má svůj vlastní operační systém a je nezávislý na operačním systému hostitele - např. můžeme mít virtuální stroje s operačními systémy Windows a Linux na jednom hostiteli.

2.1.2 Hypervizor

Hypervizor je software pro vytváření a běh virtuálních strojů. Hypervizor, někdy nazývaný ***virtual machine monitor***, izoluje operační systém hosta a prostředky virtuálních strojů a povoluje jejich tvorbu a správu.

Hypervizor zpracovává prostředky systému (CPU, paměť, úložiště) jako ***pool***, který lze snadno přerozdělit již mezi stávající virtuální stroje nebo pro tvorbu nových. Každý hypervizor potřebuje pro běh VM některé součásti na úrovni operačního systému - např. správce paměti,

plánovač procesů, I/O stack, ovladače zařízení, správce zabezpečení, síťový stack a další. Každému virtuálnímu stroji poskytuje systémové prostředky a řídí plánování zdrojů virtuálního stroje proti fyzickým prostředkům hostitele. Fyzický hardware však ale stále provádí samotné operace - procesor stále vykonává instrukce požadované virtuálním strojem, zatímco hypervizor řídí plánování.

2.1.3 Rozdělení hypervizorů

Typ 1 (nativní) - Hypervizor typu 1 běží přímo na hardwaru hostitele, kde řídí a monitoruje běh virtuálních strojů z hlediska fyzických prostředků. Hostovaný operační systém tak funguje na úrovni pod hypervizorem. Tento typ je nejběžnější v datových centrech či jiných serverových prostředích. Příklady tohoto typu jsou KVM, Microsoft Hyper-V a VMware vSphere.

Typ 2 (hostovaný) - Hypervizor typu 2 je spuštěn na konvenčním prostředí operačního systému jako softwarová vrstva nebo aplikace. Funguje tak, že abstrahuje hostované operační systémy z hostitelského operačního systému. Prostředky virtuálního stroje jsou plánovány proti hostitelskému operačnímu systému, který je poté spuštěn proti hardwaru. Hypervizor typu 2 je lepší pro jednotlivé uživatele, kteří chtějí na osobním počítači provozovat více operačních systémů. Příklady jsou Oracle VirtualBox a VMware Workstation.

Hypervizor sleduje a hlídá zdroje fyzického počítače. Když virtuální stroje spouštějí aplikace, hypervizor předává prostředky z fyzického počítače na přislušný virtuální stroj. Hypervizoři mají ovšem své vlastní režie, což znamená, že fyzické počítač musí vyčlenit určity výpočetní výkon a prostředky, aby mohla běžet samotná instance hypervizoru. To může ovlivnit celkový výkon počítače a zpomalit aplikace.

2.1.4 Paravirtualizace

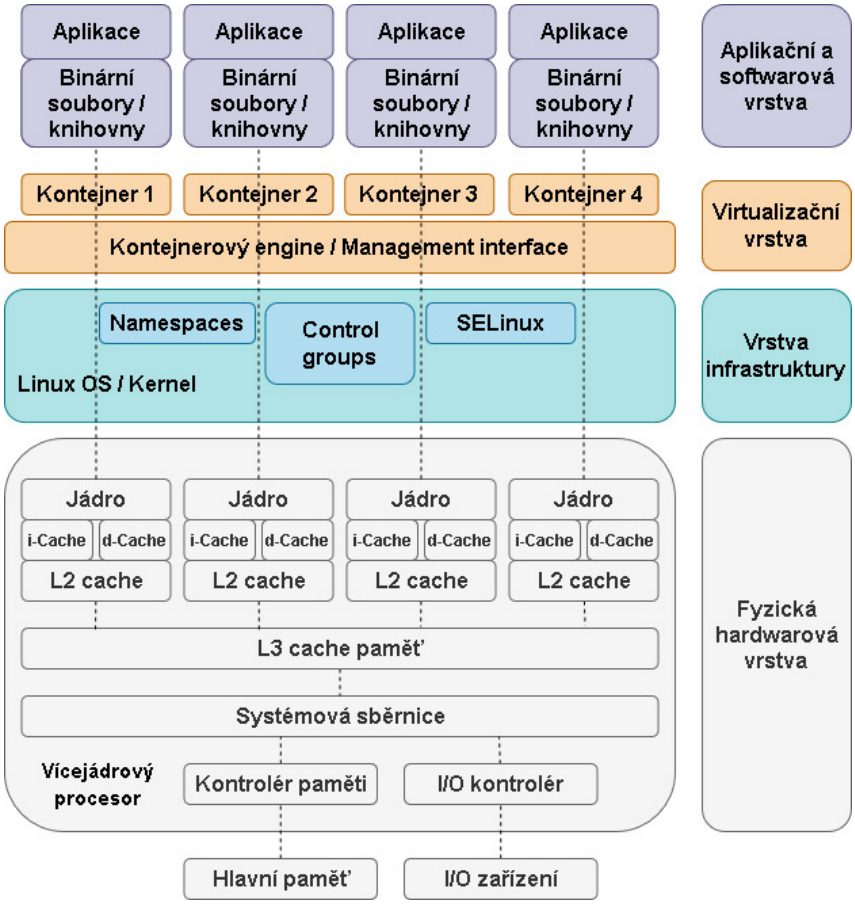
Přístup paravirtualizace je trochu odlišný. Oproti způsobu plné virtualizace si hostované systémy uvědomují přítomnost ostatních. Hypervizor zde nepotřebuje tolik výpočetního výkonu pro správu virtuálních strojů, protože jednotlivé stroje si již jsou vědomy požadavků, které ostatní stroje kladou na fyzický počítač. Celý systém tak pracuje společně jako soudržná jednotka.

2.1.5 Kontejnerová virtualizace

Kontejnery, stejně jako klasické virtuální stroje, poskytují způsob jak izolovat aplikace a poskytnout virtuální prostředí, kde mohou běžet aplikace. Existuji dva hlavní rozdíly mezi kontejnery a hypervizorovým systémem.

Kontejnerový systém pod sebou vyžaduje operační systém, který poskytuje základní služby všem kontejnerovým aplikacím tím, že pro izolaci využívá podporu virtuální paměti. Na druhé straně, hypervizor provozuje virtuální stroje, které mají svůj vlastní operační systém, využitím

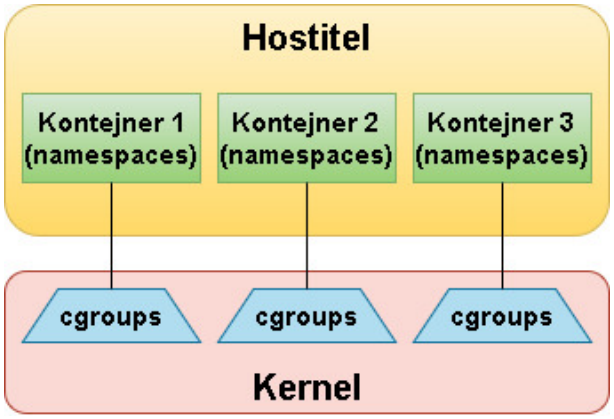
podpory fyzického hardwaru. Kontejnery mají zpravidla nižší režii než klasický virtuální stroj a typicky se zaměřují na prostředí, kde se nasazuje velké množství kontejnerů, stovky až tisíce. kontejnerové systémy také obvykle zajišťují izolaci služeb mezi kontejnery. Výsledkem je, že kontejnerové služby, jako jsou např. souborové systémy nebo síť, mohou mít omezený přístup k prostředkům.



Obrázek 1: Architektura kontejnerové virtualizace

3 Komponenty kontejnerové virtualizace

Hlavními prvky linuxové kontejnerové virtualizace jsou ***cgroups*** a ***namespaces***. Jednoduše řečeno nám cgroups umožňuje určit, kolik čeho můžeme použít a namespaces nám umožňují samotnou izolaci procesů.



Obrázek 2: Zjednodušená struktura kontejneru

3.1 Control groups

3.1.1 Co jsou to control groups?

Control groups, obvykle označované jako ***cgroups***, jsou funkcí linuxového jádra, která umožňuje organizovat procesy do hierarchických skupin, které je pak možné dále monitorovat a omezovat jejich využívání různých typů prostředků. Rozhraní cgroups je poskytováno prostřednictvím pseudosouborového systému zvaného ***cgroupfs***. Seskupování je implementováno v kódu kernelu, zatímco monitorování a omezování prostředků je implementováno v subsystému daného prostředku (paměť, CPU, atd.).[**9**]

3.1.2 Terminologie

cgroups je množina procesů, které jsou vázány na sadu omezení nebo parametrů definovaných prostřednictvím daného cgroup souborového systému.

Subsystém je komponentou jádra, která se stará o chování úloh v dané cgroupě. Byly implementovány různé typy subsystémů, které umoňují provádat takové činnosti, jako je omezení množství paměti či CPU time dostupné pro danou cgroup, účtování použitého CPU time danou cgroup a zmražení a obnovení procesů v cgroup. Subsystémy jsou někdy také známé jako kontroléry prostředků, nebo jen jednoduše kontroléry.

Hierarchie je množina cgroups, uspořádaných do stromu tak, že každá úloha v systému je přesně v jedné ze cgroups v hierarchii, a množina subsystémů; každý subsystém má svůj

systémově specifický stav přiřazený ke každé cgroup v hierarchii. Každá hierarchie má instanci virtuálního souborového systému.[**10**]

V jednu chvíli může existovat vícero aktivních hierarchií obsahujících úlohy cgroups. Každá hierarchie je oddílem všech úloh v systému.

Kód spuštěný v uživatelském prostoru může vytvářet a mazat cgroups podle jména v instanci virtuálního souborového systému, určovat a dotazovat se, ke které cgroup je úloha přiřazena, a vypsat seznam všech PID úloh přiřazených k cgroup. Tato vytváření a přiřazování ovlivňují pouze hierarchii spojenou s danou intsancí cgroup souborového systému.

Samy o sobě mají cgroup jediné využití a tím je prosté sledování úloh. Záměrem je, aby se ostatní subsystémy připojily do podpory generické cgroup, aby poskytly nové atributy, jako například účtování či omezování systémových prostředků, které mohou procesy využívat. Například, ***cpusets*** umožňují přiřadit sadu CPU a množství paměťových uzlů k úlohám v každé cgroup.

cgroups jsou pro daný kontrolér uspořádány hierarchicky. Tato hierarchie je definována vytvářením, mazáním a přejmenováním podadresářů v daném cgroup souborovém systému. Na každé úrovni hierarchie lze definovat různé atributy (např. omezování).

3.1.3 cgroups verze 1 a verze 2

Prvotní vydání implementace group bylo v jádře Linuxu verze 2.6.24. Postupem času byly přidávány různé kontroléry cgroups, aby umožnily správu zdrojů. Avšak vývoj těchto kontrolérů byl do značné míry nekoordinovaný, což mělo za následek, že mezi kontroléry vzniklo mnoho nesrovnalostí a řízení cgroup hierarchií se stalo poměrně složitým.

Kvůli problémům s počateční implementací (verze 1), byly (počínaje Linuxem 3.10) zahájeny práce na nové implementaci k odstranění těchto problémů. Původně označená jako experimentální a skrytá, nová verze (verze 2) byla nakonec oficiálně vydaná s Linuxem 4.5.

Ačkoli je cgroups verze 2 zamýšleno jako náhrada za verzi 1, starší systém stále existuje a z důvodu kompatibility nebude s velkou pravděpodobností odstraněn. V současné době cgroups v2 implementuje pouze podmnožinu kontrolerů dostupných v cgroups v1. Obě verze jsou implementovány tak, že kontroléry obou verzí mohou být připojeny na stejný systém. Je například možné použít kontroléry, které jsou podporovány ve verzi 2, zatímco také používat kontroléry verze 1, které momentálně verze 2 nepodporuje. Jedinou podmínkou je, že kontrolér nemůže být současně použit v hierarchii cgroups v1 a v hierarchii cgroups v2.

3.2 cgroups verze 1

V rámci cgroup v1 může být každý kontrolér připojen k sepárátnímu cgroup souborovému systému, který poskytuje svou vlastní hierarchickou organizaci procesů v systému. Je také možné spojit více kontrolérů (,či dokonce všechny,) proti stejnému souborovému systému, což znamená, že spojené kontroléry spravují stejnou hierarchickou organizaci procesů.

Pro každou připojenou hierarchii platí, že strom adresářů zrcadlí hierarchii cgroupy. Každá cgroup je reprezentována adresářem, přičemž každá z jejich podřízených cgroup je reprezentována jako podřízený adresář. Například /user/student/1.session reprezentuje cgroup 1.session, což je potomek cgroupy student, která je potomek /user. V každém adresáři cgroupy je množina souborů, které lze číst nebo do nich zapisovat, což poté reflektuje omezování zdrojů a několik obecných vlasností cgroups.

3.2.1 Úlohy (vlákna) a procesy

V cgroups verze 1 se rozlišuje mezi procesy a úlohami. V tomto pohledu se proces může skládat z vícero úloh (běžně nazývaných jako vlákna). Je možné nezávisle manipulovat s členstvím vláken v procesu.

Schopnost dělit vlákna napříč různými cgroupami způsobovala v některých případech problémy. Například to nedávalo smysl pro kontrolér paměti, protože všechna vlákna procesu sdílejí jediný společný adresní prostor. Kvůli těmto problémům byla v počátační implementaci cgroups verze 2 odstraněna schopnost samostatně manipulovat členství vláken v procesu a následně byla obnovena v mnohem omezenější formě.

3.2.2 Připojování kontrolérů cgroups v1

Využívání cgroups vyžaduje, aby byl kernel vytvořen s možností CONFIG_CGROUP. Kromě toho má každý z kontrolérů přiřazenou možnost konfigurace, který musí být nastavena, aby se tento kontrolér mohl použít.

Aby bylo možné kontrolér použít, musí být připojen na cgroup souborový systém. Obvyklé místo pro takové připojovací body je pod tmpfs souborových systémem připojeným v ***/sys/fs/cgroup***. Například připojení CPU kontroléru by se dalo udělat následujícím způsobem:

```
mount -t cgroup -o cpu none /sys/fs/cgroup/cpu
```

Výpis 1: Připojení CPU kontroléru

Je možné spojit více kontrolérů proti stejné hierarchii. Zde jsou například kontroléry ***cpu*** a ***cpuacct*** připojeny spolu na stejnou hierarchii:

```
mount -t cgroup -o cpu,cpuacct none /sys/fs/cgroup/cpu,cpuacct
```

Výpis 2: Připojení cpu a cpuacct kontrolérů

Připojování kontrolérů má za následek to, že proces je ve stejné cgroup pro všechny spolu připojené kontroléry. Samostatné připojení umožňuje procesu, aby byl v cgroup ***/foo1*** pro jeden kontrolér, zatímco je v cgroup ***/foo2*** pro jiný.

Je možné připojit všechny kontroléry proti stejné hierarchii.

```
mount -t cgroup -o all cgroup /sys/fs/cgroup
```

Výpis 3: Připojení všech kontrolérů cgroups v1

Stejného výsledku lze dosáhnout vynecháním parametru ***-o all***, protože to je výchozí nastavení, pokud nic nespecifikuje.

Není možné připojit stejný kontrolér na vícero cgroup hierarchií. Například, není možné připojit jak cpu, tak cpuacct kontroléry na jednu hierarchii, a zároveň cpu kontrolér na hierarchii druhou. Je možné vytvořit více přípojných bodů se stejnou sadou připojených kontrolérů. V tomto případě je však výsledkem vícero přípojných bodů poskytujích pohled na stejnou hierarchii.

Je dobré si všimnout, že v mnoha systémech jsou kontroléry cgroups v1 automaticky připojovány na ***/sys/fs/cgroup***; zejména ***systemd*** vytváří automaticky takové připojné body.

3.2.3 Odpojení kontrolérů cgroups v1

Připojený cgroup souborový systém lze odpojit pomocí příkazou ***umount***, například takto:

```
umount /sys/fs/cgroup/pids
```

Výpis 4: Odpojení cgroup souborového systému

Důležitý je však fakt, že cgroup filesystém může být odpojen pouze pokud není zaneprázdněn. To znamená, že nemá žádné cgroup potomky. Pokud tomu tak není, pak se pouze stane to, že po provedení příkazu umount se přípojný bod skryje. Aby se tedy zajistilo, že bude přípojný bod skutečné odstraněn, je potřeba nejprve odstranit všechny podřízené cgroup, což lze provést až poté, co byly všechny členské procesy přesunutý z těchto cgroup do kořenové cgroup.

3.2.4 Kontroléry cgroups v1

Všechny kontroléry jsou řízeny konfiguračním nastavením kernelu. Dodatečně, dostupnost funkce cgroups se navíc řídí volbou CONFIG_CGROUPS v nastavení kernelu.

cpu (od kernelu 2.6.24; CONFIG_CGROUP_SCHED)
: Cgroups může být zaručen minimální počet tzv. "CPU share", když je systém zaneprázdněn. To neomezuje využití CPU cgroup, pokud nejsou CPU zaneprázdněny. V kernelu verze 3.2 byl tento kontrolér rozšířen tak, aby umožňoval kontrolu nad šířkou pásma CPU. Pokud je kernel nakonfigurován s CONFIG_CFS_BANDWIDTH, pak je možné v rámci každé plánovací periody, definované v souboru v adresáři cgroups, definovat horní limit tzv. CPU time, který je přidělen procesům v cgroup. Tato horní hranice platí, i když momentálně procesor nevyužívá žádný jiný proces.

cpuacct (od kernelu v. 2.6.24; CONFIG_CGROUP_CPUACCT)
: Tento kontrolér umožňuje účtování využití CPU skupinami procesů.

cpuset (od kernelu v. 2.6.24; CONFIG_CPUSET)
Tato cgroup může být použita k propojení procesů v cgroup k specifikované sadě CPU a NUMA uzlů.

memory (od kernelu v. 2.6.25; CONFIG_MEMCG)
Kontrolér paměti podporuje hlášení a omezování paměti procesům, paměti kernelu a odkládacího prostoru využívaného cgroup.

devices (od kernelu v. 2.6.26; CONFIG_CGROUP_DEVICE) Pomocí tohoto kontroléro můžeme řídit, které procesy mohou vytvářet (mknod) zařízení a následně je otevírat pro čtení nebo zápis. Pravidla mohou být zadány jako seznamy povolených a zakázaných. Hierarchie je vynucená, což znamená, že nová pravidla nesmí porušovat již vytvořená pravidla pro cílové skupiny nebo skupiny předků.

freezer (od kernelu v. 2.6.28; CONFIG_CGROUP)
Freezer cgroup může pozastavit a znovu obnovit všechny procesy v croup. Je dobré vědět, že zastavením dané cgroup ovšem zastavíme i všechny její podřazené potomky.

net_cls (od kernel v. 2.6.29; CONFIG_CGROUP_NET_CLASSID)
Toto umístí classid, specifikovaný dle cgroup, na síťové pakety vytvořené danou cgroup. Tyto classid mohou být použity k řízení provozu pomocí nástroje tc. Toto ovšem platí pouze pro pakety opouštějící cgroup, nikoli pakety do ní přicházející.

blkio (od kernelu v. 2.6.33; CONFIG_BLK_CGROUP)
Blkio cgroup řídí a omezuje přístup k určitým blokovým zařízením tím, že ovládá zápis a čtení nastavením horních mezí a přiškrcením listů a mezilehlých uzlů v hierarchii úložiště.

perf_event (od kernelu v. 2.6.39; CONFIG_CGROUP_PERF)
Tento kontrolér umožňuje monitorování výkonu množiny procesů seskupených v cgroup.

net_prio (od kernelu v. 3.3; CONFIG_CGROUP_NET_PRIO)
Umožňuje nastavit priority pro cgroups dle síťového rozhraní.

pids (od kernelu v. 4.3; CONFIG_CGROUP_PIDS)
Umožňuje omezit počet procesů, které mohou být v cgroup vytvořeny a jejich potomků.

3.3 Namespaces

Jak již bylo řečeno, namespaces umožňují samotnou izolaci procesů a jsou nezbytnou součástí pro implementaci kontejnerů. Namespaces zabalí globální systémové prostředky do abstrakní vrstvy, díky níž se procesům uvnitř zdá, že mají svou vlastní globální instanci systémových prostředků. Změny globálních zdrojů jsou viditelné ostatním procesům, které jsou členy daného namespace, ale skryté pro ostatní procesy.[**11**]

3.3.1 Podrobnosti implementace

Kernel přiřazuje každému procesu jeden symlink na jeden druh namespace, který se nachází v ***/proc/<pid>/ns/***. Číslo inodu, na které odkazuje tento symbolický odkaz, je stejné pro každý proces v tomto namespace. Toto jednoznačně identifikuje každý namespace podle čísla inodu, na které odkazuje jeden z jeho symlinků.

Čtení symlinku pomocí nástroje readlink vrátí řetezec obsahující název druhu namespace a číslo inodu daného namespace.

Syscalls

Existují 3 systémová volání, které slouží pro práci s namespaces:

- **clone** - specifikuje, do kterého nového namespace by měl být nový proces přesunut
- **unshare** - umožňuje procesu oddělit části jeho prováděcího kontextu, které jsou momentálně sdíleny s jinými procesy
- **setns** - vstupuje do namespace specifikovaného v deskriptoru souboru

Odstranění

Pokud již není na namespace dále odkazované, tak je odstraněn. Na tom, jak se s obsaženými prostředky naloží, závisí na druhu namespace. Na namespace se dá odkazovat těmito způsoby:

1. procesem patřícím do namespace
2. otevřeným souborovým deskriptorem na namespace (/proc/<pid>/ns/<typ_namespace>)
3. pomocí připojení namespace souboru s parametrem ***bind*** (/proc/<pid>/ns/<typ_namespace>)

3.3.2 Typy namespaces

Od kernelu v. 5.6 existuje na 8 různých namespaces. Jejich funkčnost je u všech stejná; každý proces je propojen k namespace a může vidět nebo používat pouze ty prostředky, které jsou s tímto namespace spojené. Tímto způsobem může mít každý proces, nebo skupina procesů, svůj jedinečný pohled na systémové prostředky. Který prostředek je izolován závisí na druhu namespace, který byl pro danou skupinu procesů vytvořen.

Process ID (pid)

PID namespace poskytuje procesům množinu identifikátorů, který je nezávislá na ostatních namespaces. Tyto namespaces jsou vnořeny, což znamená, že nově vytvořený proces bude mít PID pro každý namespace; od toho, v němž se proces zrovna nachází, přes všechny jeho

předchůdce až po počáteční PID namespace. To znamená, že počáteční PID namespace je schopen vidět všechny procesy, i když s různými PID, než jak je vidí ostatní.

Prvnímu procesu v PID namespace je přideleno číslo 1 a dostává se mu většiny speciálního zacházení, jako má klasicky init proces, zejména to, že jsou k němu připojeny všechny osiřelé procesy v daném namespace. Toto má také za následek, že pokud je tento proces ukončen, ukončí se také všechny ostatní procesy v daném namespace, samotný namespace a všichni jeho potomci.

Network (net)

Network namespace virtualizuje síťový stack. Při vytvoření nového namespace obsahuje pouze loopback rozhraní.

Každé síťové rozhraní, ať už fyzické nebo virtuální, je přitomné pouze a právě v jednom namespace a lze jej mezi nimi přesouvat.

Každý namespace má svou privátní množinu IP adres, vlastní routovací tabulku, seznam socketů, tabulku pro sledování připojení, firewall a další prvky týkající se síťování.

Smazání namespace odstraní všechna virtuální rozhrání, která se v něm nacházejí a přesune všechna fyzická rozhraní do původního namespace.

Mount (mnt)

Mount namespace řídí přípojné body. Při vytvoření jsou přípojné body ze stávajícího namespace překopírovány do nového, ale body vytvořené až poté se již dále mezi nimi nepropagují. Pokud by bylo potřeba, je možné je mezi nimi šířit pomocí sdílených podstromů.

Přiznak pro vytvoření nového namespace tohoto typu je CLONE_NEWNS. Tento název však dostatečně nevypovídá o tom, jaký druh namespace bude vytvořen. Je to z toho důvodu, že mount namespaces byly prvním typem a vývojáři nepředpokládali, že budou existovat další.

User ID (user)

Funkce user namespace poskytují izolaci uživatelských oprávnění a segregaci identifikací uživatelů napříč vícero množinami procesů. Díky tomuto je možné vytvořit kontejner se zdánlivými administrátorskými právy, aniž by se ve skutečnosti uživatelským procesům zvýšená práva poskytla. Stejně jako PID namespace, jsou tyto namespaces vnořeny a každý nový namespace je brán jako potomek toho, jež jej vytvořil.

User namespace obsahuje mapovací tabulku, který převadí ID z pohledu kontejneru do systému. To umožňuje, aby uživatel root, přítomný v kontejneru, měl user id 0, ale v systému měl například user id například 1000000. Podobná tabulka se používá pro mapování group id.

UNIX Time Sharing (UTS)

UTS namespace umožňuje jednomu systému, aby vypadal, že má různé hostname a domain name.

Interprocess Communication (IPC)

IPC namespaces izolují procesy od meziprocesové komunikace, která je na způsob SysV. To zabraňuje procesům v různých IPC namespaces, aby například sada funkcí SHM vytvořila prostor sdílené paměti mezi dvěma procesy. Místo toho bude každý proces schopen použít stejné identifikátory pro oblast sdílené paměti a vytvořit dvě takové odlišné oblasti.

Control group (cgroup) namespace

Tento namespace skrývá identitu, kterého procesu je cgroup členem.

4 Použité technologie kontejnerové virtualizace

Přestože existuje celá řada nástrojů, poskytujících kontejnerovou virtualizaci, tak jsem si pro práci vybral nástroj LXC, respektive jeho nástavbu; LXD.

4.1 LXC

LXC, plným názvem Linux Containers, je metoda pro virtualizaci na úrovni operačního systému pro běh izolovaných Linuxových systémů (kontejnerů) na jednom hostiteli. Neposkytuje plný virtuální stroj, ale poskytuje virtuální prostředí, které má své vlastní CPU, paměť, úložiště, síť, atd. V podstatě je to uživateleské rozhraní pro funkce kernelu za účelem izolace. To je poskytnuto právě pomocí cgroup a namespaces, běžících v kernelu hostitele. Prostřednictvím API a jednoduchých nástrojů umožňuje uživatelům vytvářet a spravovat kontejnery.[**15**]

LXC kontejnery mohou být nakonfigurovány jako privilegované nebo neprivilegované. Obecně lze říci, že neprivilegované kontejnery jsou bezpečnější než privilegované, protože neprivilegované mají větši úroveň izolace díky tomu, jak jsou navrženy. Klíčovým faktorem je mapovaní UID root uživatele v kontejneru na UID non-root uživatele na hostiteli, což stěžuje případným útočníkům uvnitř kontejneru v provádění útoků na hostitelský systém. Stručně řečeno, pokud by se útočníkovi přece jen podařilo proniknout z kontejneru, měl by se na hostiteli ocitnout s velmi omezenými nebo žádnými právy.[**13**]

4.2 LXD

LXD je kontejnerový manažer nové generace. Uživatelům dává pocit, že pracují s plnými virtuálními stroji, přičemž ve skutečnosti používají kontejnery.

Je založen na práci s obrazy s širokou škálou předem připravených obrazů Linuxových distribucí a je postaven na jednoduchém, ale velice výkoném REST API.

Projekt LXD byl založen a v současné době je i veden společností Canonical Ltd.

Základem LXD je privilegovaný daemon, který vysvatuje REST API přes lokální unixový soket, popřípadě i přes síť, pokud je to povoleno. Klienti, jako například nástroj příkazové řádky poskytnutý LXD, pak dělají všechnu práci právě prostřednictím tohoto REST API. To znamená, že ať už pracujete na lokálníím hostiteli nebo vzdáleném, tak se vše chová stejně.[**16**]

4.2.1 Vztah LXD k LXC

Je důležité si uvědomit, že LXD není náhradou LXC. Ve skutečnosti je LXD nástavbou na LXD a poskytuje nový a lepší uživatelský zážitek. Pod povrchem LXD využívá LXC skrze knihovnu liblxc k vytváření a správě kontejnerů.

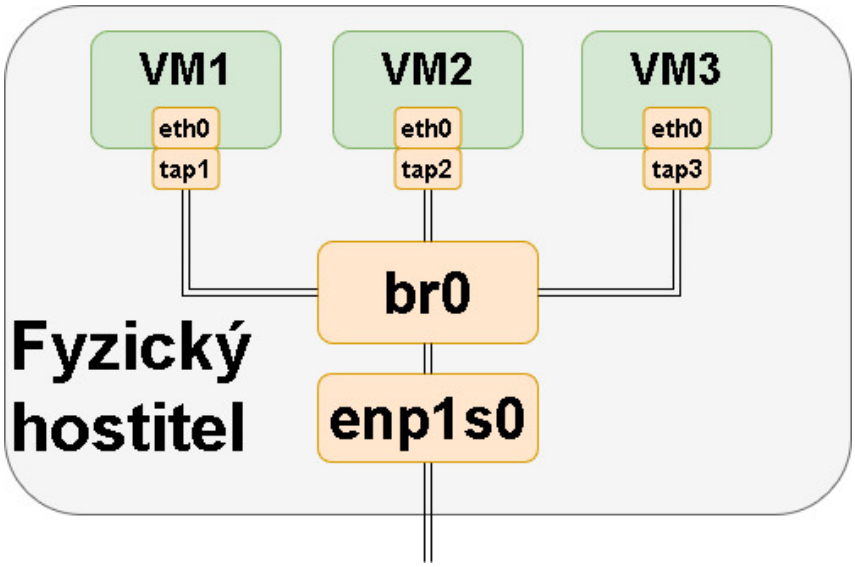
V podstatě jde o alternativu k LXC nástrojům a distribučním šablonám systému s přidanými funkcemi, která může být ovládaná i po síti.

5 Linuxová rozhraní pro virtuální síťování

Linux má bohaté možnosti virtuálního síťování, které se používají jako základ pro hostování virtuálních strojů a kontejnerů. [**5**]

5.1 Bridge(Most)

Linuxový bridge se chová jako síťový přepínač. Předává pakety mezi rozhraními, která jsou k němu připojena. Obvykle se používá pro předávání paketů na směrovačích, na branách (gateway) nebo mezi VM a síťovými jmennými prostory na hostiteli. Podporují Spanning Tree Protocol, Virtual Local Area Network (VLAN) a multicast snooping.



Obrázek 3: Schéma bridge

Bridge se hodí použít pokud potřebujeme sestavit komunikaci mezi VM, kontejnery a hostitely a dá se vytvořit takto:

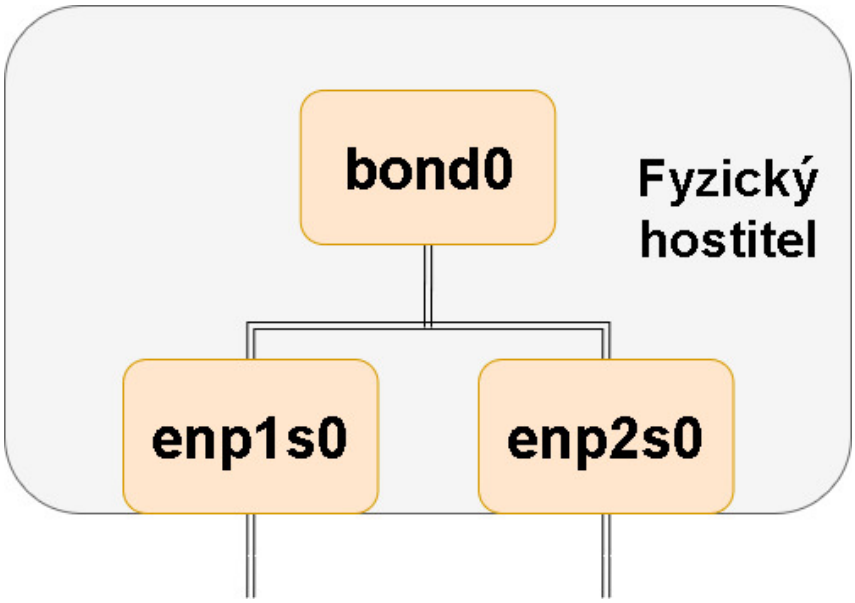
```
# ip link add br0 type bridge
# ip link set enp1s0 master br0
# ip link set tap1 master br0
# ip link set tap2 master br0
# ip link set veth1 master br0
```

Výpis 5: Vytvoření bridge

Tímto vytvoříme zařízení typu bridge br0 a přidáme mu dvě TAP zařízení (tap1, tap2), zařízení VETH (veth1) a fyzické zařízení (enp1s0), které budou sloužit jako slave.

5.2 Propojené rozhraní (Bonded interface)

Linuxový ovladač pro bonding poskytuje možnost pro agregaci vícero síťových rozhraní do jednoho logického "propojeného". Chování tohoto zařízení závisí na módu v jakém má pracovat; obecně řečeno buď jako horká záloha (hot standby) nebo služba pro vyvažování zátěže (load balancing).



Obrázek 4: Schéma bonded zařízení

Tento typ rozhraní můžeme pokud potřebujeme zvýšit rychlost linky nebo provést failover na serveru.

```
# ip link add bond0 type bond miimon 100 mode active-backup
# ip link set eth0 master bond1
# ip link set eth1 master bond1
```

Výpis 6: Vytvoření Bonded rozhraní

Konfigurace výše vytvoří bonded rozhraní nazvané bond0 v režimu active-backup.

5.3 VLAN

VLAN, neboli virtuální LAN, odděluje všesměrové domény přidáním značek do síťových paketů a umožňuje správcům sítě seskupit hostitele do stejného přepínače nebo mezi různými přepínači.

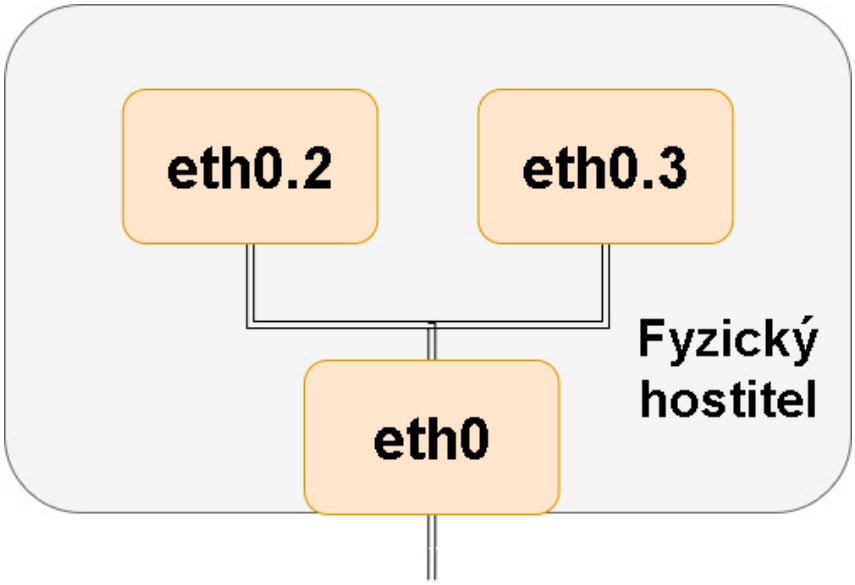
VLAN má využití, pokud je potřeba oddělit podsítě v namespace, virtuálních strojích nebo hostitelích.

Vytvořit se dá následovně:

```
# ip link add link eth0 name eth0.2 type vlan id 2
```

```
# ip link add link eth0 name eth0.3 type vlan id 3
```

Výpis 7: Vytvoření VLAN podrozhraní



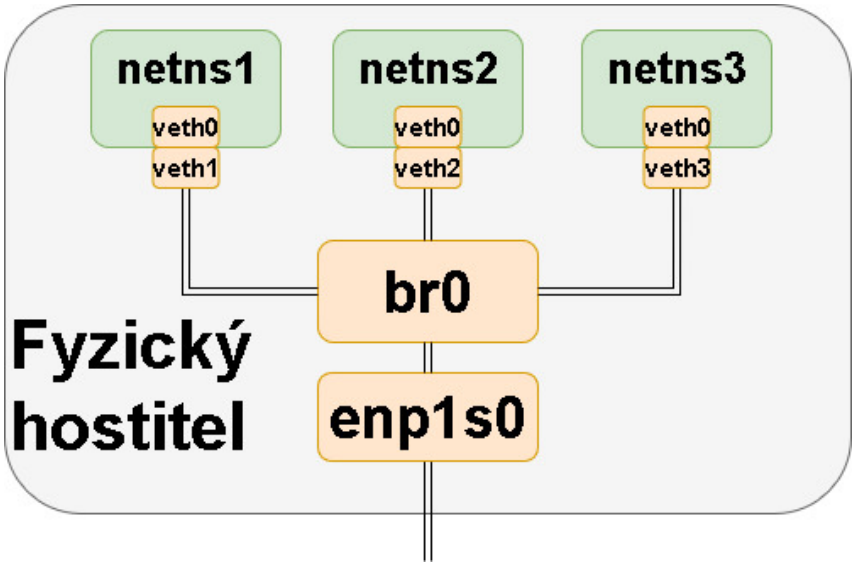
Obrázek 5: Schéma VLAN

Konfigurace vytvoří 2 rozhraní eth0.2 s VLAN značkou 2 a eth0.3 s VLAN značkou 3, jejichž rodičem je rozhraní eth0. Při konfiguraci VLAN je potřeba se ujistit, že připojený přepínač zvládá práci s VLAN značkami.

5.4 MACVLAN

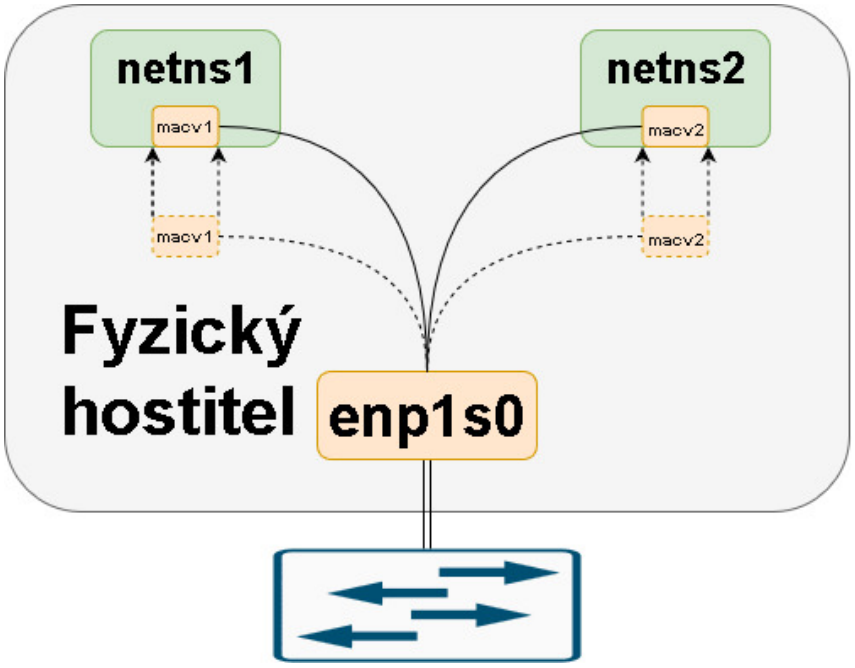
Pomocí VLAN jsme schopni vytvořit několik rozhraní nad jedním skutečným a filtrovat pakety podle značky VLAN. V režímu MACVLAN můžeme vytvořit více rozhraní s různými MAC adresami nad jedním skutečným rozhraním.

Před MACVLAN, pokud by bylo potřeba připojit VM nebo namespace k fyzické síti, musely by se vytvořit TAP/VETH zařízení, propojit je od VM k bridge a zároveň ještě k bridge připojit fyzické rozhraní, tak jak je na obrázku níže.



Obrázek 6: Schéma bridge

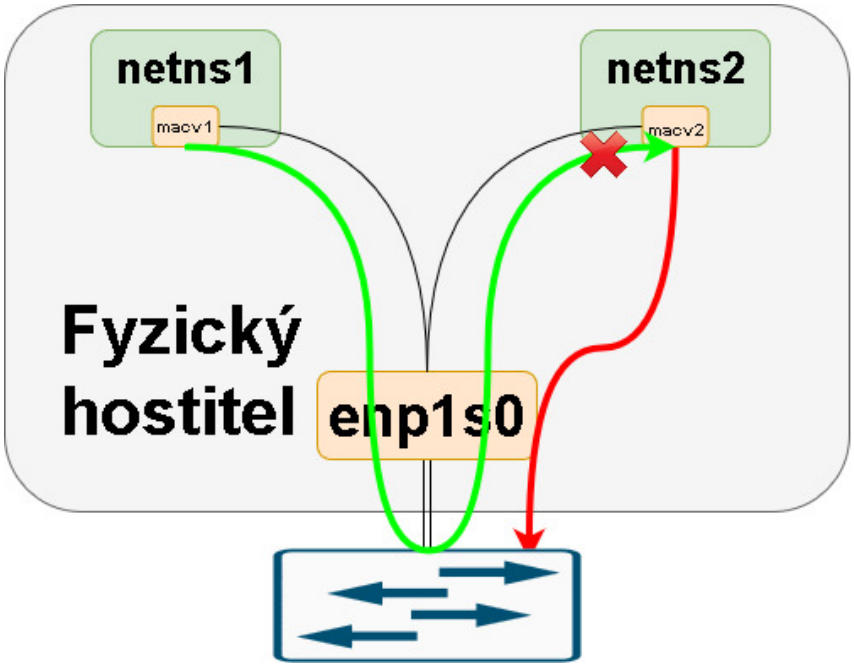
Díky MACVLAN však můžeme svázat fyzické rozhraní, které je nastaveno v režimu MACVLAN, napřímo s namespace, bez nutnosti využití bridge.



Obrázek 7: Schéma MACVLAN

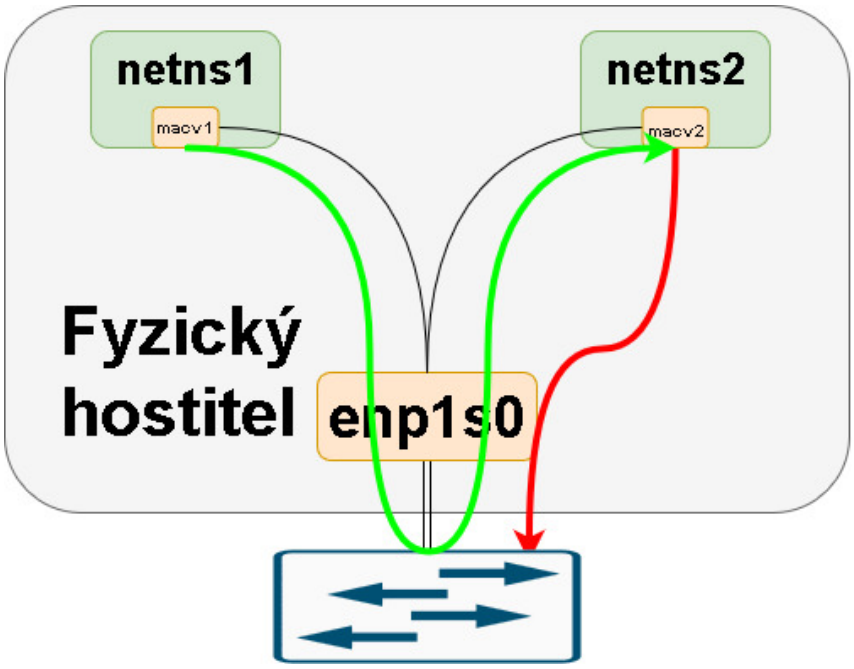
Existuje několik typů MACVLAN, mezi nejznámější patří tyto:

1. **Private**: privátní režim neumožňuje komunikaci mezi instancemi MACVLAN, které jsou vytvořené nad stejným fyzickým rozhraním, ani pokud externí přepínač podporuje režim hairpinning.



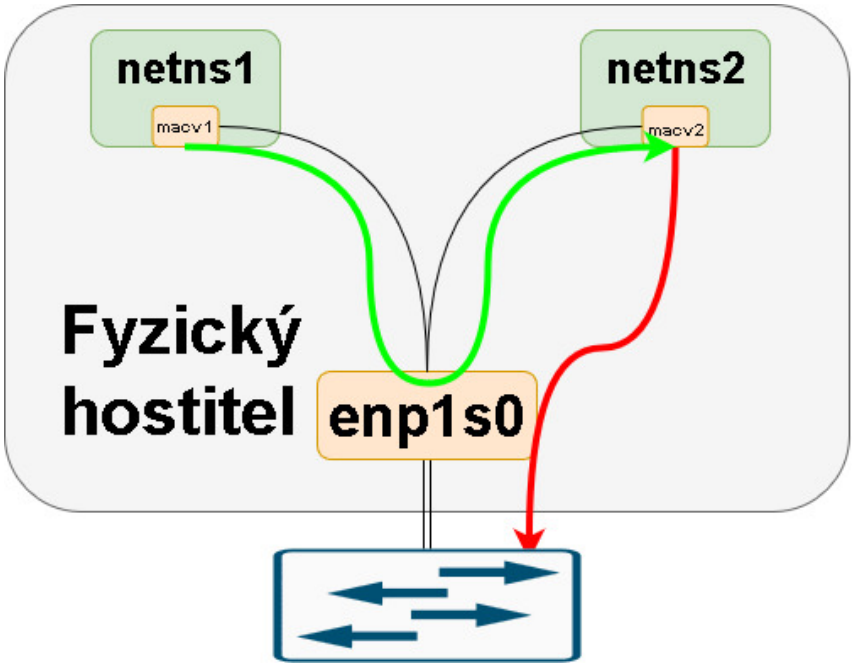
Obrázek 8: Schéma MACVLAN v režimu Private

2. **VEPA**: data jsou přenášena z jedné instance MACVLAN do druhé jsou přenášena přes fyzické rozhraní.



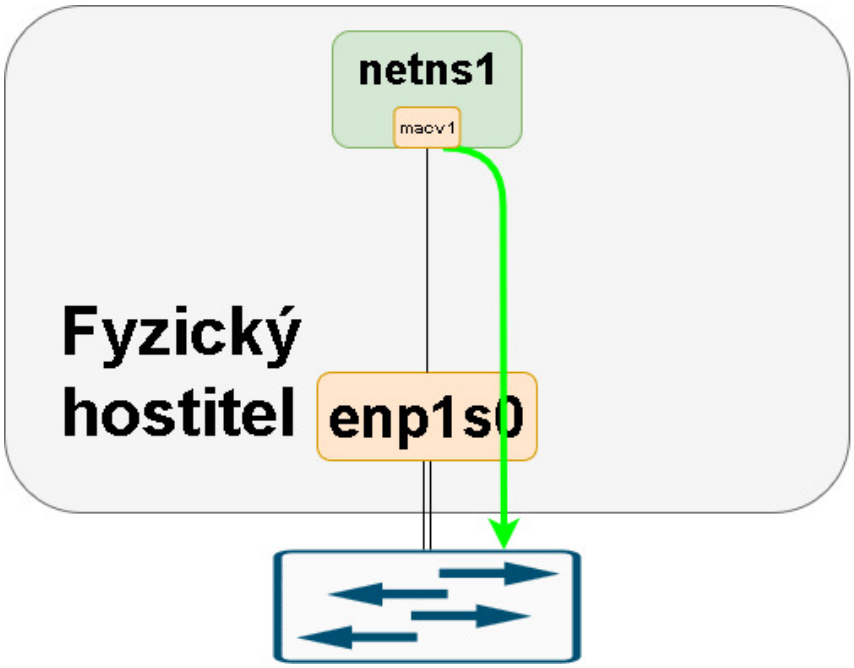
Obrázek 9: Schéma MACVLAN v režimu VEPA

3. **Bridge**: všechny koncové body jsou přímo propojeny pomocí jednoduchého bridge přes fyzické rozhraní.



Obrázek 10: Schéma MACVLAN v režimu Bridge

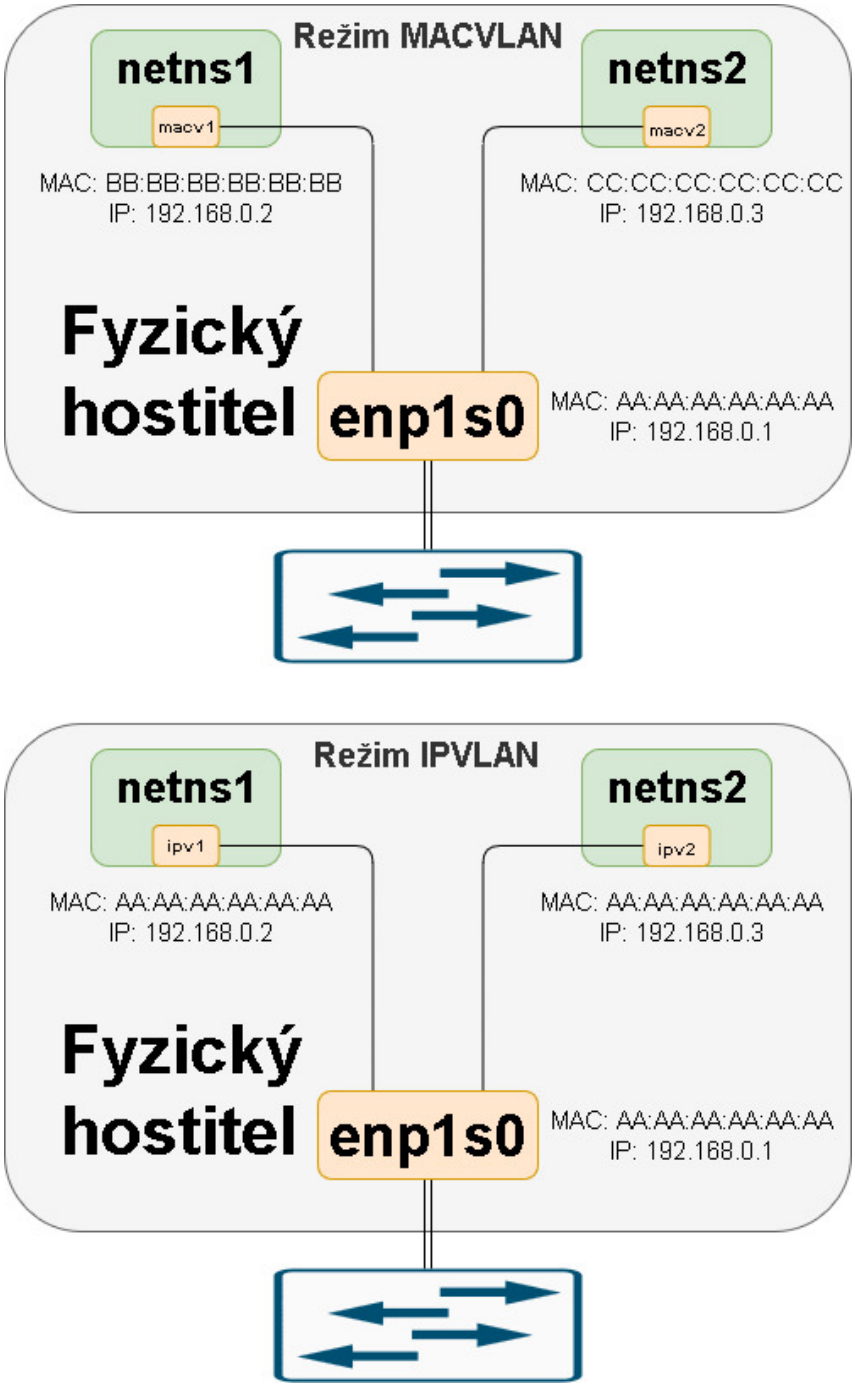
4. **Passthru**: umožňuje přímé připojení jednoho virtuálního stroje k fyzickému rozhraní.



Obrázek 11: Schéma MACVLAN v režimu Passthru

5.5 IPVLAN

Režim IPVLAN je podobný jako MACVLAN s tím rozdílem, že koncové body mají stejnou MAC adresu.



Obrázek 12: MACVLAN vs IPVLAN

IPVLAN podporuje režimy L2 a L3. IPVLAN L2 se chová jako MACVLAN typu bridge.

Rodičovským rozhraním je v tom případě bridge nebo přepínač. V IPVLAN L3 je rodičovským rozhraním směrovač a pakety jsou mezi koncovými body směrovány, což dává lepší škálovatelnost. Režim IPVLAN by se měl použít v těchto případech:

a Pokud je rodičovské rozhraní bezdrátové

b Výkon rodičovského rozhraní klesa, protože se překročil počet různých MAC adres

c Fyzický přepínač limituje počet MAC adres povolených na portu - Port Security

V každém jiném případě se doporučuje použít režim MACVLAN.

IPVLAN se dá nastavit takto:

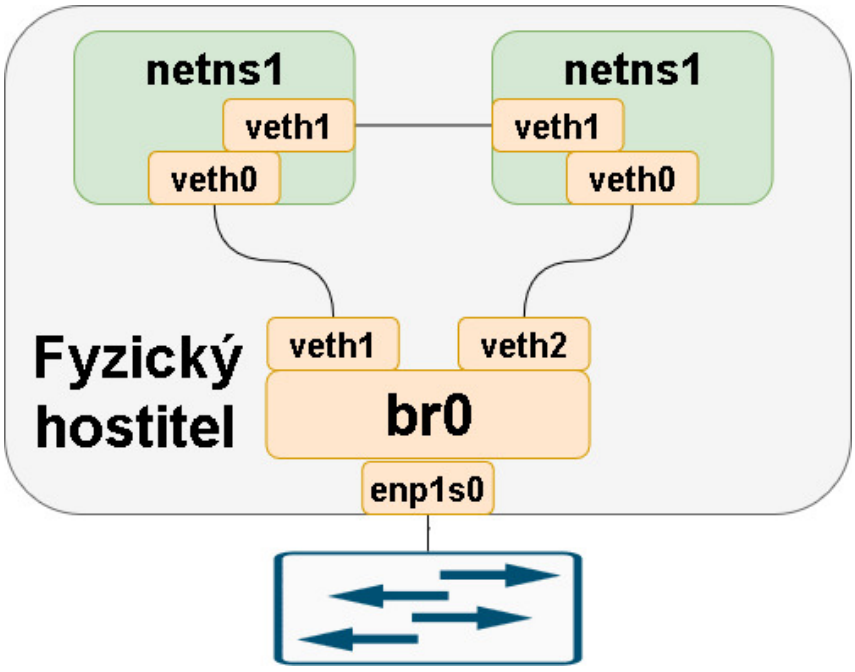
```
# ip netns add ns0
# ip link add name ipvl0 link eth0 type ipvlan mode l2
# ip link set dev ipvl0 netns ns0
```

Výpis 8: Vytvoření IPVLAN

5.6 VETH

Zařízení VETH se chová jako lokální Ethernetový tunel. Zařízení jsou vytvářena v párech, jak ukazuje obrázek níže.

Pakety přenášené na jednom zařízení v páru jsou okamžitě přijatá na druhém zařízení. Pokud je kterékoli zařízení z páru vypnuté, celé spojení je také přerušeno.



Obrázek 13: Schéma VETH zařízení

VETH zařízení využijeme tehdy, pokud je potřeba komunikovat mezi namespace virtuálního stroje a globálním prostředím hostitele, nebo virtuální stroje potřebují komunikovat napřímo mezi sebou.

VETH zařízení se dají nakonfigurovat takto:

```
# ip netns add net1
# ip netns add net2
# ip link add veth1 netns net1 type veth peer name veth2 netns net2
```

Výpis 9: Vytvoření VETH

Konfigurace výše vytvoří 2 namespace, net1 a net2, a pár VETH zařízení, veth1 a veth2. Veth1 se přiřadí do namespace net1 a veth2 do net2. To daná namespaces propojí a pokud se na veth zařízení přiřadí IP adresy, tak bude možné mezi nimi komunikovat.

6 Praktická část

Pro praktickou část diplomové práce jsem využil školní počítače s operačním systémem Ubuntu 18.04 LTS. Obsahují čtyřjádrový procesor a 16 GB operační paměti, kde 8 GB je použito jako klasická operační paměť a dalších 8 GB jako disk, tzn. při bootování se operační systém uložený na SSD, který slouží pouze jako read-only, nahraje do operační paměti, kde se s ním pak pracuje dále. Při rebootu či vypnutí stroje jsou veškerá data ztracena a při startu se nahraje opět čistý systém z SSD.

Všechny přikazy jsou prováděny pod uživatelem root.

6.1 Instalace a prvotní konfigurace nástroje LXD

Po nabootování počítače je dobré jako první aktualizovat repozitáře balíčků, ze kterých budeme instalovat, a upgradovat stávající nainstalované programy. Následně můžeme nainstalovat i samotný nástroj LXD.

```
# apt update -y && apt upgrade -y && apt install -y lxd
```

Výpis 10: Instalace nástroje LXD

Po stáhnutí a nainstalování nástroje je potřeba spustit inicializační proces *lxd init*, kde se nastaví základní konfigurace, aby bylo možné s nástrojem pracovat. Celý proces a konkrétní hodnoty jsou ukázány na obrázku 14.

Jako první jsme dotázání, jestli chceme použít LXD clustering. Pokud zvolíme možnost ano, je potřeba zvolit, jaký bude jeho název uzlu v clusteru, přes jakou IP adresu se s ním bude komunikovat, zda se bude připojovat do již existujícího clusteru, nebo se teprve bude vytvářet a heslo, vůči kterému se budou uzly ověřovat. Pro naše potřeby však clustering využívat nebudeme.

Následuje nastavení storage poolu, kam se budou jednotlivé kontejnery ukládat. Můžeme vytvořit nový, či použít již vytvořený. V systému Ubuntu se standardně nacházejí v adresáři ***/var/lib/lxd/storage-pools/***, což se však může u jiných distribucí lišit. Vytvoříme nový pool se jménem ***lxd_storage***.

Dále máme na výběr službu MaaS, kterou využívat nebudeme.

Poté následuje nastavení lokálního síťového bridge. Buď můžeme použít již nějaký existující, vytvořený jinými nástrojemi, jako například openvswitch či brctl, nebo vytvořit nový. Za přepokladu, že chceme vytvořit nový bridge, jsme dotázáni, jaké IP adresy chceme používat.

Na výběr jsou tyto možnosti:

1. **none** - na rozhraní bridge se žádná IP adresa nenastaví a ani samotné kontejnery žádné adresy automaticky nedostanou

2. **auto** - na rozhraní bridge se nastaví náhodná IP adresa, kontejnery poté také dostanou náhodnou adresu z daného rozsahu

- pro IPv4: náhodná adresa z rozsahu 10.0.0.0/8, poté se maska upraví na /24
- pro IPv6: náhodná adresa z rozsahu fd42::/16, poté se délka prefixu upraví na /64

3. **manual** - uživatel si sám zvolí IP adresu pro rozhraní bridge, kontejnery poté dostanou náhodnou adresu ze zvoleného rozsahu

Je dobré vědět, že pokud se u manuálního nastavení IPv6 adresy vybere prefix jiný než /64, nebudou kontejnery adresy automaticky dostávat. Využívá se zde totiž lokální *DHCP* server *dnsmasq*, který pro konfiguraci IPv6 adres využívá *SLAAC*. Novému bridge necháme výchozí název ***lxdbr0*** a přiřadíme mu IPv6 adresu ***fd00::1/64***. IPv4 protokol používat nebudeme.

Při manuálním nastavení je také potřeba povolit nebo zamítnout dynamický NAT, tzv. maškarádu. V našem případě maškarádu povolíme, aby měly kontejnery přístup do internetu, kdy se jejich adresa přeloží na adresu rozhraní hostitele, které vede do internetu. Při automatickém nastavení se NAT povolí bez dotazu. Provádí se pomocí nástroje *iptables*.

Další možnosti je možné ponechat ve výchozím nastavení; není potřeba vzdálená správa, cachování obrazů se může nechat povolené a vypsaní právě nastavené konfurace ve formátu YAML je volitelné. Není však úplně od věci si jej nechat vypsat, zda je všechno v pořádku).

```
root@pc2:~# lxd init
Would you like to use LXD clustering? (yes/no) [default=no]:
Do you want to configure a new storage pool? (yes/no) [default=yes]:
Name of the new storage pool [default=default]: lxd_storage
Would you like to connect to a MAAS server? (yes/no) [default=no]:
Would you like to create a new local network bridge? (yes/no) [default=yes]:
What should the new bridge be called? [default=lxdbr0]:
What IPv4 address should be used? (CIDR subnet notation, "auto" or "none") [default=auto]: none
What IPv6 address should be used? (CIDR subnet notation, "auto" or "none") [default=auto]: fd00::1/64
Would you like LXD to NAT IPv6 traffic on your bridge? [default=yes]:
Would you like LXD to be available over the network? (yes/no) [default=no]:
Would you like stale cached images to be updated automatically? (yes/no) [default=yes]
Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]: yes
config: {}
networks:
- config:
    ipv4.address: none
    ipv6.address: fd00::1/64
    ipv6.nat: "true"
  description: ""
  managed: false
  name: lxdbr0
  type: ""
storage_pools:
- config: {}
  description: ""
  name: lxd_storage
  driver: dir
profiles:
- config: {}
  description: ""
  devices:
    eth0:
      name: eth0
      nictype: bridged
      parent: lxdbr0
      type: nic
    root:
      path: /
      pool: lxd_storage
      type: disk
  name: default
cluster: null
```

Obrázek 14: proces lxd init

Z hlediska sítě se po dokončení procesu provede několik důležitých věcí. Vytvoří se onen bridge lxdbr0 se specifikovanou adresou fd00::1/64, vytvoří se pravidlo pro maškarádu adres fd00::/64 pomocí nástrone ip6tables a povolí se packet forwarding pro protokol IPv6. Pro protokol IPv4 se nepovolí, jelikož jsme jej při lxd init procesu nepovolili.

```
root@pc2:~# ip6tables -L POSTROUTING -t nat
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all      fd00::/64           !fd00::/64            /* generated for LXD network lxdbr0 */
root@pc2:~# ip6tables -S POSTROUTING -t nat
-P POSTROUTING ACCEPT
-A POSTROUTING -s fd00::/64 ! -d fd00::/64 -m comment --comment "generated for LXD network lxdbr0" -j
MASQUERADE
root@pc2:~# cat /proc/sys/net/ipv6/conf/all/forwarding
1
```

Obrázek 15: Ip6tables pravidla pro NATování adres

6.1.1 Vytvoření kontejnerů

Jakmile máme úspěšně dokončený inicializační proces, je možné vytvořit kontejnery.

Kontejnery s distribucí Ubuntu 18.04 vytvoříme následujícím příkazem:

```
# lxc launch ubuntu:b ub18a
# lxc launch ubuntu:b ub18b
```

Výpis 11: Vytvoření kontejnerů

Při prvním vytvoření si hostitel nejprve stáhne z úložiště obraz dané distribuce. Při druhém a každém dalším vytvoření si jen načte již stáhnutý obraz, což značně celý proces urychluje.

Do kontejnerů můžeme buď přistupovat přímo tím, že si spustíme jeho shell, anebo můžeme vykonávat jen jednotlivé příkazy přímo z hostitele pomocí dvojitého spojovníku. Ve výchozím stavu je příhlašování pomocí SSH zakázáno.

```
# lxc exec ub18a bash
# lxc exec ub18a -- "libovolný příkaz"
```

Výpis 12: Provádění příkazů v kontejneru

Kontejnery jsou vytvářeny s určitým profilem nastavení; pokud profil nespecifikujeme, vytvoří se s profilem default, který byl nastaven při procesu lxd init. Co se nastavení sítě kontejnerů týče, ve výchozím nastavení mají kontejnery síťové rozhraní eth0 typu bridged. Rodičem rozhraní je bridge lxdbr0.

IPv6 adresy jsou kontejnerům generovány pomocí SLAAC s využitím EUI-64, tzn. jsou vytvářeny na základě MAC adres kontejnerů.

6.2 Síťování v prostředí LXD

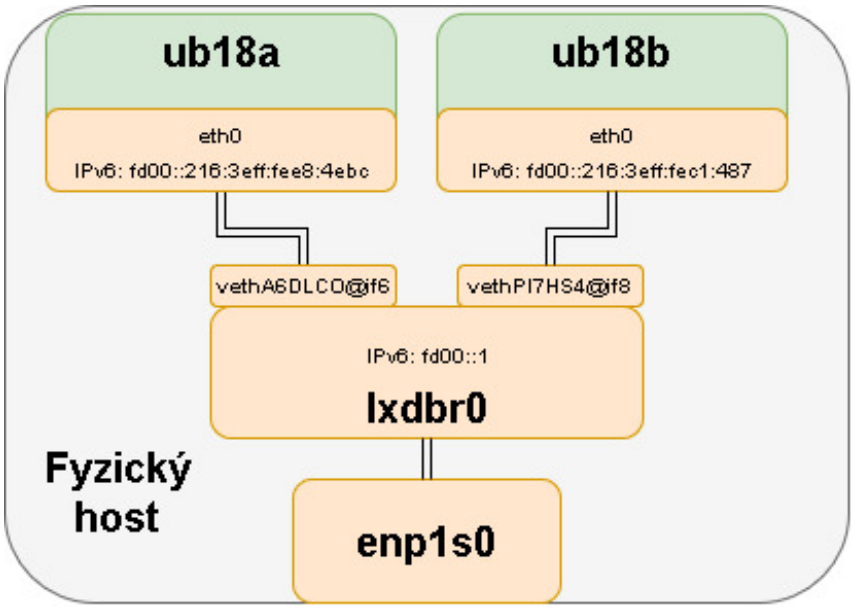
6.2.1 Bridge

Pokud necháme proces ***lxd init*** ve výchozím stavu, pouze s upravenými IP adresami, tak se nám vytvoří bridge ***lxdbr0***. Bridge se chová stejně jako klasický switch. Při vytváření kontejnerů se vytvoří pár VETH rozhraní; na straně hostitele je pojmenování jako ***veth***, šestimístný řetězec, skládající se z náhodné kombinace čísel a velkých písmen, zavináč a číslo rozhraní druhého konce tunelu. Tato rozhraní jsou podřízena rozhraní ***lxdbr0***. Můžeme si to představit jako by to byl ***lxdbr0*** fyzický switch a ***veth*** rozhraní jeho porty.

Na obrázcích 16 a 17 můžeme vidět reálné IPv6 adresy.

```
5: lxdbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether fe:10:ff:7e:68:67 brd ff:ff:ff:ff:ff:ff
    inet6 fd00::1/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::b822:4aff:fe85:2b8f/64 scope link
       valid_lft forever preferred_lft forever
7: vethA6DLCO@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master lxdbr0 state UP gro
up default qlen 1000
    link/ether fe:39:22:1f:15:45 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::fc39:22ff:fe1f:1545/64 scope link
       valid_lft forever preferred_lft forever
9: vethPI7HS4@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master lxdbr0 state UP gro
up default qlen 1000
    link/ether fe:10:ff:7e:68:67 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::fc10:ffff:fe7e:6867/64 scope link
       valid_lft forever preferred_lft forever
root@pc2:~# lxc list
+-------+---------+------+-------------------------------+------------+-----------+
| NAME  |  STATE  | IPV4 |              IPV6             |    TYPE    | SNAPSHOTS |
+-------+---------+------+-------------------------------+------------+-----------+
| ub18a | RUNNING |      | fd00::216:3eff:fee8:4ebc (eth0) | PERSISTENT | 0         |
+-------+---------+------+-------------------------------+------------+-----------+
| ub18b | RUNNING |      | fd00::216:3eff:fec1:487 (eth0)  | PERSISTENT | 0         |
+-------+---------+------+-------------------------------+------------+-----------+
```

Obrázek 16: Adresace kontejnerů v režimu Bridged



Obrázek 17: Schéma kontejnerů

Při komunikaci není potřeba, aby měl lxdbr0 IPv6 adresu, protože zde probíhá jen přepínaní na druhé vrstvě pomocí MAC adres. Je však nutné, aby rozhraní jako takové běželo. Pokud by se vyplo, komunikace bude přerušena. To stejně platí pro veth rozhraní na straně hostitele; své IPv6 adresy nepotřebují, tak jako porty switche, ale musí být zapnuté.

V tomto základním nastavení je však dobré, aby lxdbr0 měl svou adresu a kontejnery s ním byly schopné komunikovat, protože využívají fyzického hostitele jako svůj DNS server a výchozí bránu. Pokud bychom odstranili IPv6 adresu a kontejnery nebyly schopné komunikovat přímo

s hostitelem, přišli bychom o službu DNS serveru. Komunikace do internetu by mohla stále fungovat, ale muselo by se do směrovací tabulky vložit nová výchozí cesta směřující na link-local adresu lxdbr0.

Problém s DNS je možné obejít tím, že použijeme univerzitní DNS server; využívání jiných DNS serverů (mimo univerzitu) je blokováno.

```
lxc exec ub18a -- tail /etc/resolv.conf
lxc exec ub18a -- sed -i 's/127.0.0.53/2001:718:1001::53/g' /etc/resolv.conf
```

Výpis 13: Nastavení DNS serveru

Prvním příkazem si můžeme ověřit obsah souboru, který obsahuje aktuální používaný DNS server a druhým příkazem nahradíme lokální DNS za IPv6 adresu univerzitní DNS.

```
root@ub18a:~# nslookup google.com
Server:         2001:718:1001::53
Address:        2001:718:1001::53#53

Non-authoritative answer:
Name:   google.com
Address: 216.58.201.78
Name:   google.com
Address: 2a00:1450:4014:80d::200e

root@ub18a:~# ping6 google.com -c 3
PING google.com(prg03s06-in-x0e.1e100.net (2a00:1450:4014:80d::200e)) 56 data bytes
64 bytes from prg03s06-in-x0e.1e100.net (2a00:1450:4014:80d::200e): icmp_seq=1 ttl=55 time=13.6 ms
64 bytes from prg03s06-in-x0e.1e100.net (2a00:1450:4014:80d::200e): icmp_seq=2 ttl=55 time=13.6 ms
64 bytes from prg03s06-in-x0e.1e100.net (2a00:1450:4014:80d::200e): icmp_seq=3 ttl=55 time=13.6 ms

--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 13.618/13.662/13.685/0.031 ms
```

Obrázek 18: Otestování DNS serveru

6.2.2 MACVLAN

Jak již bylo zmíněno v kapitole 5.4, režim MACVLAN nám umožňuje nakonfigurovat vícero MAC adres na jedno fyzické rozhraní, což znamená, že nám umožňuje vytvořit podrozhraní z daného fyzické rozhraní, kde každé podrozhraní má svou unikátní a náhodnou MAC adresu a posléze i svou vlastní IPv6 adresu. IPv6 adresu dostane ze subnetu nadřazeného rozhraní - rodiče. kontejnery nemohou komunikovat s nadřazeným rozhraním napřímo, tzn. kontejner nemůže komunikovat s hostitelem. Pro zajištění komunikace je potřeba vytvořit další podrozhraní a přiřadit jej hostiteli.

Konfigurace se dá dělat různými způsoby. Nejjednodušším způsobem je vzít výchozí profil, zkopírovat jej do nového a upravit.

```
# lxc profile copy default profile-macvlan
# lxc profile device set profile-macvlan eth0 nictype macvlan
# lxc profile device set profile-macvlan eth0 parent enp1s0
```

Výpis 14: Vytvoření MACVLAN profilu

Můžeme zkontrolovat správné nastavení nového profilu a porovnat rozdíly s výchozím profilem default.

```
root@pc2:~# lxc profile show profile-macvlan
config: {}
description: Default LXD profile
devices:
  eth0:
    name: eth0
    nictype: macvlan
    parent: enp1s0
    type: nic
  root:
    path: /
    pool: lxd_storage
    type: disk
name: profile-macvlan
used_by: []
```

```
root@pc2:~# lxc profile show default
config: {}
description: Default LXD profile
devices:
  eth0:
    name: eth0
    nictype: bridged
    parent: lxdbr0
    type: nic
  root:
    path: /
    pool: lxd_storage
    type: disk
name: default
used_by:
- /1.0/containers/ub18a
- /1.0/containers/ub18b
```

Obrázek 19: Porovnání výchozího profilu s novým MACVLAN profilem

Nyní jen vytvoříme kontejnery s novým profilem.

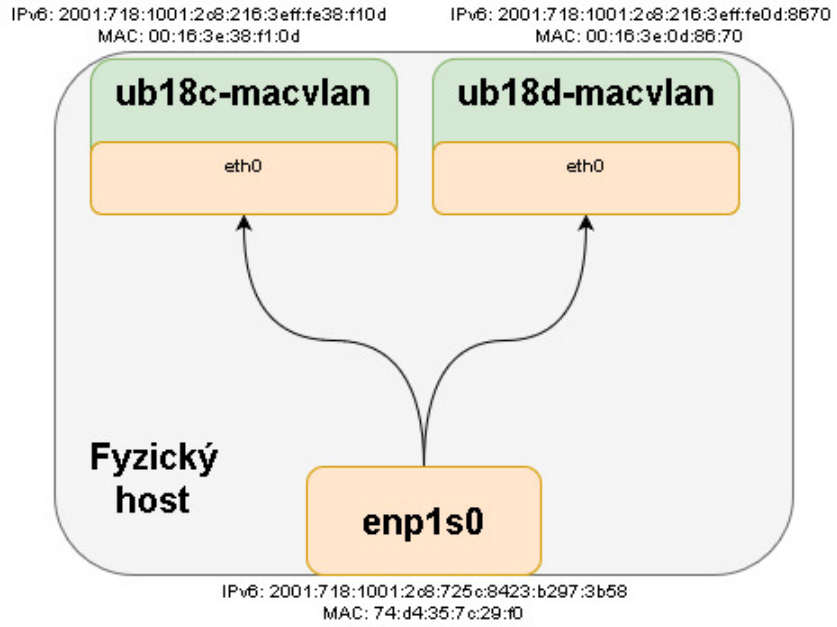
```
# lxc launch -p profile-macvlan ubuntu:b ub18c-macvlan
# lxc launch -p profile-macvlan ubuntu:b ub18d-macvlan
```

Výpis 15: Vytvoření kontejnerů v režimu MACVLAN

Když se nyní podíváme na přidělené IPv6 adresy nových kontejnerů, tak zjistíme, že dostaly IPv6 z rozsahu školní sítě 2001:718:1001:2c8::/64. Zbytek adresy, tj. interface id, je opět vygenerován pomocí EUI-64.

```
root@pc2:~# lxc list
+---------------+---------+------+-----------------------------------------------+------------+-----------+
|     NAME      |  STATE  | IPV4 |                     IPV6                      |    TYPE    | SNAPSHOTS |
+---------------+---------+------+-----------------------------------------------+------------+-----------+
| ub18a         | RUNNING |      | fd00::216:3eff:fee8:4ebc (eth0)               | PERSISTENT | 0         |
+---------------+---------+------+-----------------------------------------------+------------+-----------+
| ub18b         | RUNNING |      | fd00::216:3eff:fec1:487 (eth0)                | PERSISTENT | 0         |
+---------------+---------+------+-----------------------------------------------+------------+-----------+
| ub18c-macvlan | RUNNING |      | 2001:718:1001:2c8:216:3eff:fe38:f10d (eth0)   | PERSISTENT | 0         |
+---------------+---------+------+-----------------------------------------------+------------+-----------+
| ub18d-macvlan | RUNNING |      | 2001:718:1001:2c8:216:3eff:fe0d:8670 (eth0)   | PERSISTENT | 0         |
+---------------+---------+------+-----------------------------------------------+------------+-----------+
```

Obrázek 20: Adresace kontejnerů v režimu MACVLAN



Obrázek 21: Schéma kontejnerů v režimu MACVLAN

Nyní ovšem nastává problém, daný bezpečnostní politikou učebny. Switch, do kterého je fyzický počítač připojen, má nastavenou Port-Security omezenou pouze na MAC adresu NIC enp1s0. Z kontejnerů tedy není možné komunikovat dále, než mezi ostatními kontejnery. U režimu bridge tento problém nebyl, protože se MAC adresa při opuštění fyzického počítače změnila na povolenou MAC adresu.

6.2.3 Physical

Tento režim je v podstatě tvrdé namapování fyzického rozhraní hostitele na virtuální rozhraní kontejneru. Jakmile se vytvoří kontejner s tím profilem, celé rozhraní se okamžitě přesune do kontejneru a z hostitele zmizí. Cokoliv je pak připojeno kabelem na rozhraní již prakticky nekomunikuje s hostitelem, ale napřímo s kontejnerem.

Jako fyzické rozhraní jsem využil síťový USB adaptér. Pro naše potřeby se dá říci, že se chová stejně jako klasická integrovaná NIC. Samozřejmě, že to tak není úplně pravda (USB NIC je řízená CPU, sdílí sběrnici s dalšími USB zařízeními, atd.), ale pro naše potřeby to můžeme přehlednout.

```
root@pc2:~# ip a a fd01::1/64 dev enx00133b9cbf40
root@pc2:~# ip a s enx00133b9cbf40
4: enx00133b9cbf40: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen
1000
    link/ether 00:13:3b:9c:bf:40 brd ff:ff:ff:ff:ff:ff
    inet6 fd01::1/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::213:3bff:fe9c:bf40/64 scope link
       valid_lft forever preferred_lft forever
```

Obrázek 22: USB adaptér v globálním namespace

Konfigurace se provede obdobně jako u režimu MACVLAN, akorát se upraví nictype:

```
# lxc profile copy default profile-physical
# lxc profile device set profile-physical eth0 nictype physical
# lxc profile device set profile-physical eth0 parent enx00133b9cbf40
# lxc launch -p profile-physical ubuntu:b ub18e-physical
```

Výpis 16: Vytvoření Physical profilu

Po vytvoření se můžeme podívat, že rozhraní enx00133b9cbf40 zmizelo z hostitele a přesunulo se do kontejneru na eth0. Při tomto procesu přišlo o svou IPv6 unique-local adresu, kterou je poté potřeba nakonfigurovat znova, nicméně MAC adresa a link-local adresa zůstaly totožné.

```
root@pc2:~# lxc exec ub18e-physical -- ip a s eth0
4: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:13:3b:9c:bf:40 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::213:3bff:fe9c:bf40/64 scope link
       valid_lft forever preferred_lft forever
```

Obrázek 23: USB adaptér v kontejneru

6.2.4 VLAN

Nástroj LXD sám o sobě neumí práci s VLANy. K jejich dosažení je možné využít postup v kapitole výše, nebo zkusit použít jiný nástroj. Já jsem využil nástroj OpenVSwitch, což je open-source, vícevrstvý virtuální switch. Můžeme jej nainstalovat klasicky z repozitářů nástrojem apt a vytvořit nový bridge ovsbr0:

```
# apt install openvswitch-switch
# ovs-vsctl add-br ovsbr0
```

Výpis 17: Instalace nástroje OpenVSwitch a přidání nového bridge

Je zde však nevýhoda oproti bridge, vytvořeného nástrojem LXD, a to ta, že bridge není vytvořen jako managed.[**19**] Tzn. není možné na něm provádět některé LXD příkazy, jako například přiřazování IPv6 adresy, místo toho se musí nastavovat manuálně pomocí klasických linuxových nástrojů (ip/ifconfig), či nastavovat DNS, DHCPv6, atd.

OpenVSwitch můžeme do LXD přidat již v rámci procesu lxd init, při výběru bridge, tím pádem by se přidal do vychozího profilu default, anebo můžeme opět vytvořit nový profil a jen jej upravit. Pokud bychom jej přidávali v rámci procesu lxd init, tak je dobré vědět, že se v profilu uloží jako typ macvlan; je tedy potřeba jej změnit na bridged.

```
# lxc profile copy default profile-ovs
# lxc profile device set profile-ovs eth0 parent ovsbr0
# lxc launch -p profile-ovs ubuntu:b ub18a-ovs
# lxc launch -p profile-ovs ubuntu:b ub18b-ovs
# lxc launch -p profile-ovs ubuntu:b ub18c-ovs
```

```
# lxc launch -p profile-ovs ubuntu:b ub18d-ovs
# lxc exec ub18a-ovs -- ip a a fd02::a/64 dev eth0
# lxc exec ub18b-ovs -- ip a a fd02::b/64 dev eth0
# lxc exec ub18c-ovs -- ip a a fd02::c/64 dev eth0
# lxc exec ub18d-ovs -- ip a a fd02::d/64 dev eth0
```

Výpis 18: Vytvoření kontejnerů v rámci OpenVSwitche

Jakmile máme vytvořené kontejnery a přiřazené adresy, je možné vzít VETH rozhraní na hostiteli a přiřadit jim konkrétní VLAN značky.

```
# ovs-vsctl set port vethLAN064 tag=10
# ovs-vsctl set port veth53RTG4 tag=20
# ovs-vsctl set port vethP9I38T tag=10
# ovs-vsctl set port veth7V7C11 tag=20
//analogicky by se konfigurovalo na Cisco přepínači
switch# configure terminal
switch(config)# interface fa0/1
switch(config-if)# switchport mode access
switch(config-if)# switchport access vlan 10
```

Výpis 19: Zařazení kontejnerů do VLAN

Tímto jsme kontejnery ub18a-ovs (fd02::a) a ub18c-ovs (fd02::c) přiřadili do VLANy 10 a kontejnery ub18b-ovs (fd02::b) a ub18d-ovs (fd02::d) do VLANy 20.

```
root@pc15:~# lxc list
+-----------+---------+------+----------------+------------+-----------+
|   NAME    |  STATE  | IPV4 |      IPV6      |    TYPE    | SNAPSHOTS |
+-----------+---------+------+----------------+------------+-----------+
| ub18a-ovs | RUNNING |      | fd02::a (eth0) | PERSISTENT | 0         |
+-----------+---------+------+----------------+------------+-----------+
| ub18b-ovs | RUNNING |      | fd02::b (eth0) | PERSISTENT | 0         |
+-----------+---------+------+----------------+------------+-----------+
| ub18c-ovs | RUNNING |      | fd02::c (eth0) | PERSISTENT | 0         |
+-----------+---------+------+----------------+------------+-----------+
| ub18d-ovs | RUNNING |      | fd02::d (eth0) | PERSISTENT | 0         |
+-----------+---------+------+----------------+------------+-----------+
root@pc15:~# ovs-vsctl show
22c0fd6d-afa7-48bb-b31e-6b5ca9c0f652
    Bridge "ovsbr0"
        Port "vethP9I38T"
            tag: 10
            Interface "vethP9I38T"
        Port "vethLAN064"
            tag: 10
            Interface "vethLAN064"
        Port "veth53RTG4"
            tag: 20
            Interface "veth53RTG4"
        Port "veth7V7C11"
            tag: 20
            Interface "veth7V7C11"
        Port "ovsbr0"
            Interface "ovsbr0"
                type: internal
    ovs_version: "2.9.5"
```

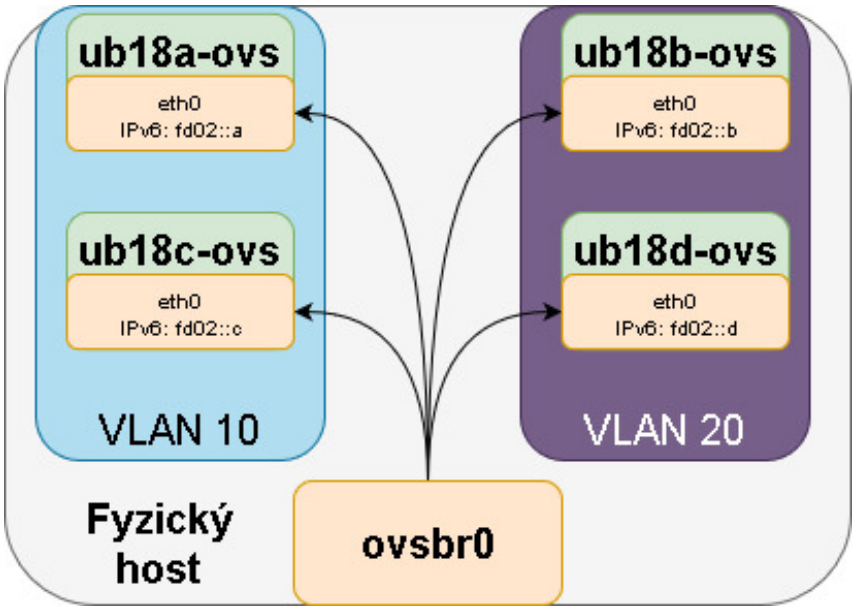
Obrázek 24: Adresace kontejnerů v rámci OpenVSwitche

Není problém otestovat, zda bude skutečně fungovat jen konektivita mezi kontejnery ve stejné VLAN. Na následujícím obrázku můžeme vidět, že kontejner ub18a-ovs má skutečně přístup jen ke kontejneru ub18c-ovs, který je ve stejné VLAN.

```
root@pc15:~# lxc exec ub18a-ovs -- ping6 fd02::b -c 3
PING fd02::b(fd02::b) 56 data bytes
From fd02::a icmp_seq=1 Destination unreachable: Address unreachable
From fd02::a icmp_seq=2 Destination unreachable: Address unreachable
From fd02::a icmp_seq=3 Destination unreachable: Address unreachable

--- fd02::b ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2034ms

root@pc15:~# lxc exec ub18a-ovs -- ping6 fd02::c -c 3
PING fd02::c(fd02::c) 56 data bytes
64 bytes from fd02::c: icmp_seq=1 ttl=64 time=0.299 ms
64 bytes from fd02::c: icmp_seq=2 ttl=64 time=0.067 ms
64 bytes from fd02::c: icmp_seq=3 ttl=64 time=0.064 ms

--- fd02::c ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2025ms
rtt min/avg/max/mdev = 0.064/0.143/0.299/0.110 ms
root@pc15:~# lxc exec ub18a-ovs -- ping6 fd02::d -c 3
PING fd02::d(fd02::d) 56 data bytes
From fd02::a icmp_seq=1 Destination unreachable: Address unreachable
From fd02::a icmp_seq=2 Destination unreachable: Address unreachable
From fd02::a icmp_seq=3 Destination unreachable: Address unreachable

--- fd02::d ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2026ms
```

Obrázek 25: Testování konektivity kontejnerů v rámci VLAN

Z výpisu je zřejmé, že konfigurace funguje správně, tak jak má.



Obrázek 26: Schéma zapojení kontejnerů ve VLAN

Není samozřejmě žádný problém vytvořit bridge v rámci LXD více, postupovalo by se stejně jako v případě jednoho.

6.2.5 Network namespace v rámci LXD

Network namespace se nám stará o izolaci síťového prostředí pro kontejnery. Na rozdíl od jiných namespace, jako například PID nebo UID, jen nepřemapovává ID na jiná, ale skutečně vytvoří celé nové prostředí.

Pokud bychom chtěli vytvořit vlastní network namespace, udělali bychom to pomocí příkazu ***ip netns add jmeno_namespace***. Tímto bychom vytvořili nový pojmenovaný namespace. Příkazem ***ip netns list*** bychom mohli vypsat všechny namespaces, které v počítači jsou a příkazem ***ip netns exec jmeno_namespace bash*** se do něj přímo přepnout a konfigurovat jej, jak bychom chtěli.

Každý vytvořený kontejner má takový svůj vlastní namespace. S tím rozdílem, že tento namespace není pojmenovaný, příkazem pro vylistování jej vidět nelze, tudíž s ním nelze pracovat. Abychom si jej mohli zobrazit, tak je to potřeba udělat menší oklikou.

Jako první jsem na ukázku vytvořil 2 kontejnery ub18a a ub18b. Pokud použijeme příkaz pro vylistování, nic se nám nezobrazí. Je možné však vylistovat namespaces podle jejich ID. Po provedení příkazu ***ip netns list-id*** můžeme vidět, že existují 2 nepojmenovaná prostředí s ID 0 a 1.

```
root@pc3:~# lxc launch ubuntu:b ub18a
Creating ub18a
Starting ub18a
root@pc3:~# lxc launch ubuntu:b ub18b
Creating ub18b
Starting ub18b
root@pc3:~# lxc list
+-------+---------+------+-----------------------------------+------------+-----------+
| NAME  |  STATE  | IPV4 |               IPV6                |    TYPE    | SNAPSHOTS |
+-------+---------+------+-----------------------------------+------------+-----------+
| ub18a | RUNNING |      | fd00::216:3eff:fe73:2b0a (eth0) | PERSISTENT | 0         |
+-------+---------+------+-----------------------------------+------------+-----------+
| ub18b | RUNNING |      | fd00::216:3eff:fe9d:65cb (eth0) | PERSISTENT | 0         |
+-------+---------+------+-----------------------------------+------------+-----------+
root@pc3:~# ip netns list
root@pc3:~# ip netns list-id
nsid 0
nsid 1
```

Obrázek 27: Namespaces kontejnerů beze jména

Nyní bychom chtěli přiřadit danému prostředí nějaké jméno, abychom do něj mohli posléze přistupovat. K tomu nám poslouží následující skript, který si nejprve z infa o kontejneru zjistí jeho ID procesu, pod kterým běží, následně vytvoří složku pro daný namespace (, pokud už složka neexistuje) a nakonec vytvoří symbolický odkaz z procesu kontejneru na danou složku. Skript se spouští s parametrem, které je název kontejneru.

```
# vim ns_name.sh

    //zacatek skriptu
    #!/bin/bash
    nazev_kontejneru=$1
    pid=$(lxc info $1 | grep Pid | awk '{print $2}')
    mkdir -p /var/run/netns
    ln -sf /proc/$pid/ns/net /var/run/netns/$nazev_kontejneru
    //konec skriptu

# chmod +x ns_name.sh
# ./ns_name ub18a
# ./ns_name ub18b
```

Výpis 20: Skript pro pojmenování namespaces kontejnerů

Jakmile si namespaces necháme vylistovat nyní, uvidíme, že již se zde objevují pod jménem názvu jejich kontejneru. Pro kontrolu si můžeme nechat vypsat IPv6 adresy, jak pomocí nástroje LXD, tak pomocí samotného nástroje ip pro práci s namespaces. V prvním případě je ub18a samozřejmě název kontejneru, v druhém případě už však je ub18a namespace.

```
root@pc3:~# lxc list
+-------+---------+------+---------------------------------+------------+-----------+
| NAME  |  STATE  | IPV4 |              IPV6               |    TYPE    | SNAPSHOTS |
+-------+---------+------+---------------------------------+------------+-----------+
| ub18a | RUNNING |      | fd00::216:3eff:fe73:2b0a (eth0) | PERSISTENT | 0         |
+-------+---------+------+---------------------------------+------------+-----------+
| ub18b | RUNNING |      | fd00::216:3eff:fe9d:65cb (eth0) | PERSISTENT | 0         |
+-------+---------+------+---------------------------------+------------+-----------+
root@pc3:~# ip netns list
ub18b (id: 1)
ub18a (id: 0)
root@pc3:~# lxc exec ub18a -- ip a | grep inet6
    inet6 ::1/128 scope host
    inet6 fd00::216:3eff:fe73:2b0a/64 scope global dynamic mngtmpaddr noprefixroute
    inet6 fe80::216:3eff:fe73:2b0a/64 scope link
root@pc3:~# ip netns exec ub18a ip a | grep inet6
    inet6 ::1/128 scope host
    inet6 fd00::216:3eff:fe73:2b0a/64 scope global dynamic mngtmpaddr noprefixroute
    inet6 fe80::216:3eff:fe73:2b0a/64 scope link
```

Obrázek 28: Porovnání adresace mezi namespace a kontejnerem

6.2.6 cgroups a omezování prostředků v rámci LXD

LXD nám umožňuje kontrolu a omezování prostředků, které jim hostitel poskytuje. Velikost místa na disku, přístup k procesoru, velikost operační paměti, rychlost zapisování a čtení disku a v neposlední řadě rychlost síťového rozhraní ve vstupním i výstupním směru.

Co se týče omezování rychlosti sítě, stačí upravit profil požadovaného kontejneru. Je zde však podmínka, že omezené rozhraní musí být typu bridged nebo p2p. Nelze tedy omezovat rozhraní typu macvlan či physical a jiné.

Otestovat můžeme například pomocí nástroje iperf3. Nainstalujeme jej na hostitele i kontejner a následně kontejner spustíme v režimu server a hostitele, jež bude kontejner testovat, v režimu klienta.

Na obrázku 29 můžeme vidět, že přenosová šířka pásma byla mezi cca 26 Gbit/s až 37 Gbit/s, tzn. nebylo zde žádné omezení.

```
root@pc15:~# lxc list
+-------+---------+------+--------------------------------+------------+-----------+
| NAME  |  STATE  | IPV4 |              IPV6              |    TYPE    | SNAPSHOTS |
+-------+---------+------+--------------------------------+------------+-----------+
| ub18a | RUNNING |      | fd00::216:3eff:fedf:9f67 (eth0) | PERSISTENT | 0         |
+-------+---------+------+--------------------------------+------------+-----------+
root@pc15:~# iperf3 -c fd00::216:3eff:fedf:9f67
Connecting to host fd00::216:3eff:fedf:9f67, port 5201
[  4] local fd00::1 port 52506 connected to fd00::216:3eff:fedf:9f67 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  3.14 GBytes  26.9 Gbits/sec    0    402 KBytes
[  4]   1.00-2.00   sec  3.30 GBytes  28.4 Gbits/sec    0    402 KBytes
[  4]   2.00-3.00   sec  2.75 GBytes  23.6 Gbits/sec    0    527 KBytes
[  4]   3.00-4.00   sec  4.13 GBytes  35.5 Gbits/sec    0    527 KBytes
[  4]   4.00-5.00   sec  4.31 GBytes  37.0 Gbits/sec    0    527 KBytes
[  4]   5.00-6.00   sec  4.33 GBytes  37.2 Gbits/sec    0    527 KBytes
[  4]   6.00-7.00   sec  3.58 GBytes  30.8 Gbits/sec    0    770 KBytes
[  4]   7.00-8.00   sec  3.44 GBytes  29.6 Gbits/sec    0    770 KBytes
[  4]   8.00-9.00   sec  3.32 GBytes  28.5 Gbits/sec    0    770 KBytes
[  4]   9.00-10.00  sec  3.05 GBytes  26.2 Gbits/sec    0    770 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  35.3 GBytes  30.4 Gbits/sec    0             sender
[  4]   0.00-10.00  sec  35.3 GBytes  30.4 Gbits/sec                  receiver

iperf Done.
```

Obrázek 29: Šířka pásma bez omezení

Pokud bychom chtěli šířku pásma omezit, na například 100 Mbit/s, stačilo by provést tyto příkazy.

```
# lxc profile device set default eth0 limits.ingress 100Mbit
# lxc profile device set default eth0 limits.ingress 100Mbit
```

Výpis 21: Omezení rychlosti na 100 Mbit/s

Tímto omezíme rychlost vstupní a výstupní rychlosti rozhraní eth0, který se nachází v profilu default. Ověřit si to můžeme na obrázku 30, kde vidíme, že šířka pásma se opravdu zmenšila na oněch 100Mbit/s a s tím klesla i velikost přenesených dat.

```
root@pc15:~# iperf3 -c fd00::216:3eff:fedf:9f67
Connecting to host fd00::216:3eff:fedf:9f67, port 5201
[  4] local fd00::1 port 52514 connected to fd00::216:3eff:fedf:9f67 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  12.0 MBytes   100 Mbits/sec    0    113 KBytes
[  4]   1.00-2.00   sec  11.2 MBytes  94.1 Mbits/sec    0    113 KBytes
[  4]   2.00-3.00   sec  11.3 MBytes  94.6 Mbits/sec    0    113 KBytes
[  4]   3.00-4.00   sec  11.2 MBytes  94.1 Mbits/sec    0    113 KBytes
[  4]   4.00-5.00   sec  11.3 MBytes  94.6 Mbits/sec    0    113 KBytes
[  4]   5.00-6.00   sec  11.2 MBytes  94.1 Mbits/sec    0    113 KBytes
[  4]   6.00-7.00   sec  11.2 MBytes  94.1 Mbits/sec    0    113 KBytes
[  4]   7.00-8.00   sec  11.3 MBytes  94.6 Mbits/sec    0    113 KBytes
[  4]   8.00-9.00   sec  11.2 MBytes  94.1 Mbits/sec    0    113 KBytes
[  4]   9.00-10.00  sec  11.3 MBytes  94.6 Mbits/sec    0    113 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec   113 MBytes  94.9 Mbits/sec    0             sender
[  4]   0.00-10.00  sec   113 MBytes  94.4 Mbits/sec                  receiver

iperf Done.
```

Obrázek 30: Šířka pásma s omezením na 100 Mbit/s

Bohužel se mi nepovedlo provoz omezit pomocí samotných cgroups. Je zde možné pouze značkovat síťové pakety pomocí net_cls a pomocí nástroje tc přiřadit jednotlivým značkám prioritu.

Co se ale například týče omezení paměti, zde už to je zajímavější. Ve výchozím stavu není kontejner nijak omezený a má přístup k plné paměti, tak jako jeho hostitel.

```
root@pc15:/sys/fs/cgroup/memory/lxc/ub18a# lxc exec ub18a -- free -h
              total        used        free      shared  buff/cache   available
Mem:           6.8G         46M        6.8G        284K        284K        6.8G
Swap:            0B          0B          0B
root@pc15:/sys/fs/cgroup/memory/lxc/ub18a# pwd; cat memory.limit_in_bytes
/sys/fs/cgroup/memory/lxc/ub18a
9223372036854771712
root@pc15:/sys/fs/cgroup/memory/lxc/ub18a# lxc config set ub18a limits.memory 256MB
root@pc15:/sys/fs/cgroup/memory/lxc/ub18a# lxc exec ub18a -- free -h
              total        used        free      shared  buff/cache   available
Mem:           244M         46M        197M        284K        284K        197M
Swap:            0B          0B          0B
root@pc15:/sys/fs/cgroup/memory/lxc/ub18a# pwd; cat memory.limit_in_bytes
/sys/fs/cgroup/memory/lxc/ub18a
256000000
root@pc15:/sys/fs/cgroup/memory/lxc/ub18a# echo 500000000 > memory.limit_in_bytes
root@pc15:/sys/fs/cgroup/memory/lxc/ub18a# lxc exec ub18a -- free -h
              total        used        free      shared  buff/cache   available
Mem:           476M         46M        430M        284K        284K        430M
Swap:            0B          0B          0B
root@pc15:/sys/fs/cgroup/memory/lxc/ub18a# pwd; cat memory.limit_in_bytes
/sys/fs/cgroup/memory/lxc/ub18a
499998720
```

Obrázek 31: Omezování operační paměti kontejneru

Na obrázku 31 vidíme, že kontejner má bez omezení 6,8 GB paměti, stejně jako hostitel. Stejnou hodnotu můžeme vidět v cgroup souboru ***memory.limit_in_bytes***, který se nachází v adresáři ***/sys/fs/cgroup/memory/lxc/ub18a/***. Jakmile paměť omezíme pomocí LXD, změna se projeví jak v kontejneru, tak v daném souboru. Pokud natvrdo přepíšeme hodnotu v souboru, změna se také projeví v kontejneru. Jakmile však dojde k rebootu kontejneru, hodnota se opět přepíše na hodnotu, kterou jsme specifikovali pomocí LXD.

7 Závěr

Podstatou a hlavním cílem této práce bylo pochopit problematiku kontejnerové virtualizace a možnosti síťování v ní. V práci jsem podrobně popsal kontejnerovou virtualizaci, její komponenty, možností síťování v prostředí Linuxu a poté samotné možností síťování v kontejnerovém prostředí LXD s použitím protokolu IPv6. Pokusil jsem co nejlépe a nejsrozumitelněji popsat jednotlivé typy síťových rozhraní a síťování použité v daném prostředí.

V práci jsem se snažil vysvětlit, jak funguje kontejnerová virtualizace pod povrchem. Tzn. nesnažil jsem se jen ukázat, jak dané konfigurace nastavit pomocí příkazů, ale snažil jsem se i ukázat, jak k samotné izolaci a rozdělování prostředků dochází pomocí cgroups a namespaces. Dále jsem se snažil vysvětlit, co z hlediska síťování v LXD lze a co nelze, popřípadě jak to obejít a pomoci si jinými nástroji, které zvládnou to, co LXD nezvládne.

Tuto diplomovou práci bych doporučil lidem, které tyto technologie zajímají a chtěli by s němi pracovat. Virtualizace je totiž v nynější době dá se říci nutností, protože z vlastní zkušenosti vím, že valná většina serverů v produkčním prostředí jsou virtuály.

Jako další vývoj této práce bych rád viděl pokročilé možnosti clusteringu a migrace kontejnerů při výpadku uzlů v prostředí LXD, jelikož redundance je v dnešní době nutností a není možné si dovolit ztrátu dat.

Odkazy

[1] GUNARATNE, Imesh. The Evolution of Linux Containers and Their Future: A history of containerization technology starting in 1979, and what the future holds for Docker and similar technologies. DZone: Cloud Zone [online]. 2016-07-18. Dostupné z: https://dzone.com/articles/evolution-of-linux-containers-future

[2] A Brief History of Containers: Learn how containers and microservices are changing the face of software architecture, development, and operations. D2IQ [online]. 2018-07-19. Dostupné z: https://d2iq.com/blog/brief-history-containers

[3] MARQUEZ, Ell. The History of Container Technology. Linux Academy [online]. 2018-08-17. Dostupné z: https://linuxacademy.com/blog/linux-academy/history-of-container-technology/

[4] OSNAT, Rani. A Brief History of Containers: From the 1970s Till Now. Aqua: Container Security, Serverless Security and Cloud Native Security [online]. 2020-01-10. Dostupné z: https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

[5] FIRESMITH, Donald. Virtualization via Containers. Carnegie Mellon University: Software Engineering Institute [online]. 2017-09-25. Dostupné z: https://insights.sei.cmu.edu/sei_blog/2017/09/virtualization-via-containers.html

[6] CHAMBERLAIN, Doug. Containers vs. Virtual Machines (VMs): What's the Difference? NetApp [online]. 2018-03-16. Dostupné z: https://blog.netapp.com/blogs/containers-vs-vms/

[7] G WONG, William. What's the Difference Between Containers and Virtual Machines? Electronic Design [online]. 2016-07-15. Dostupné z: https://www.electronicdesign.com/technologies/dev-tools/article/21801722/whats-the-difference-between-containers-and-virtual-machines

[8] POLLOCK, Antonia. Virtualization vs. Containerization. Liquid Web [online]. 2019-12-24. Dostupné z: https://www.liquidweb.com/kb/virtualization-vs-containerization/

[9] MENAGE, Paul. CGROUPS. The Linux Kernel Archives [online]. 2018-02-11. Dostupné z: https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

[10] Cgroups - Linux control groups. Linux Manual Pages [online]. Dostupné z: http://man7.org/linux/man-pages/man7/cgroups.7.html

[11] Namespaces - overview of Linux namespaces. Linux Manual Pages [online]. Dostupné z: http://man7.org/linux/man-pages/man7/namespaces.7.html

[12] Linux Containers. ArchLinux [online]. Dostupné z: https://wiki.archlinux.org/index.php/Linux_Contain

[13] LXD. ArchLinux [online]. Dostupné z: https://wiki.archlinux.org/index.php/LXD

[14] Linux Containers Forum [online]. Dostupné z: https://discuss.linuxcontainers.org/

[15] What's LXC? Linux Containers: LXC Introduction [online]. Dostupné z: https://linuxcontainers.org/lxc/introduction/

[16] What's LDC? Linux Containers: LXD Introduction [online]. Dostupné z: https://linuxcontainers.org/lxd/introduction/

[17] LXD - system container manager: Network Configuration. LXD Docs [online]. Dostupné z: https://lxd.readthedocs.io/en/latest/networks/

[18] HANGBIN, Liue. Introduction to Linux interfaces for virtual networking. Red Hat: Developers [online]. 2018-10-22. Dostupné z: https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking/

[19] Only managed networks can be modified. Linux Container: Forum [online]. 2018-9-13. Dostupné z: https://discuss.linuxcontainers.org/t/error-only-managed-networks-can-be-modified/2665